UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY


Proyecto de Grado

# Web Honeypot

*Autores:*
Federico Nicolás Pernas
Javier Agustín Sanchez
Nicolás Zeballos

*Supervisores:*
Gustavo Betarte
Rodrigo Martinez
Marcelo Rodríguez

2020-12-27

**ABSTRACT**

Las Aplicaciones Web suponen un gran atractivo para las Organizaciones, ya que mediante el desarrollo de una única aplicación web se puede llegar a un mayor público, debido a que éstos pueden acceder a ella fácilmente mediante, por ejemplo, un navegador. Sin embargo, el crecimiento de estas aplicaciones traen consigo preocupaciones sobre su seguridad.

En los últimos años se ha estudiado como incorporar Honeypots, los cuales son dispositivos de seguridad (hardware o software) encargados de engañar a los atacantes, dentro del ambito de las Aplicaciones Web. Estos dispositivos son altamente empleados a nivel de seguridad de la red, pero es deseable poder aplicar sus conceptos y principios para proteger dichas Aplicaciones Web. A partir de estas premisas es que este proyecto se basa en la investigación sobre la posibilidad de usar un Honeypot para proteger Aplicacioes Web en un ambiente de producción sin alterar su operativa.

De manera de cumplir este objetivo, se especifica un framework genérico y extensible, el cual se puede ver como un Honeypot Embebido de Aplicaciones Web que funciona dentro de la misma aplicación. El Honeypot hará uso de Honeytokens para atraer la atención de los atacantes y poder detectar su posible intrusión analizando el estado de estos componentes en los distintos mensajes intercambiados en la comunicación.

Keywords: Honeypot, Honeytoken, Aplicaciones Web, HTTP, Framework, extensible.

# Índice

# 1. Introducción

La seguridad informática tomó una trascendencia en ascenso constante desde el incremento del uso de las aplicaciones web. La gran mayoría manejan información sensible sobre los usuarios, motivo por el cual los atacantes realizan diversas acciones intrusivas sobre las aplicaciones web en búsqueda de vulnerabilidades para hacerse poseedores de esa información. Asimismo, estos atacantes también pueden interesarse en dañar directamente a la aplicación mediante ataques conocidos, como por ejemplo denegación de servicio.

Existen diversas formas de reconocer problemas de seguridad sobre las aplicaciones web, una de ellas es realizar múltiples pruebas de seguridad, tarea conocida como *hacking ético*[1]. Su finalidad está en identificar vulnerabilidades sobre determinada aplicación, para lo que es debido situarse del lado atacante con el objetivo de reconocer como perjudicar a dicha aplicación. A su vez, existe otro enfoque más proactivo, el cual consiste en agregar dispositivos de seguridad a la infraestructura, como son los Honeypot. Estos Honeypots son utilizados para atraer a los atacantes hacia ellos en búsqueda de que sus ataques no logren vulnerar a los activos principales, y además generar información sobre los ataques recibidos. Sin embargo, la concepción de los Honeypots no fue enfocada en el ámbito de las aplicaciones web, por lo que el trabajo hasta el momento acerca de estos dispositivos en esta área es muy poco. Por lo tanto:

*En este proyecto se persigue el objetivo de investigar la adaptación de Honeypot dentro del área de las aplicaciones web en ambientes de producción, evitando un impacto significativo sobre la arquitectura establecida.*

Este documento se distribuye de la siguiente forma: en la sección Revisión de antecedentes se presenta el contexto teórico a partir del documento *Estado del Arte* [1]. En la sección Análisis y Diseño se presenta un resumen del documento *Análisis y diseño* [2] donde se definen nuevos conceptos a partir de un análisis del comportamiento y el funcionamiento que debe tener un Honeypot como el deseado. En la sección Implementación se describe y desarrolla un caso de estudio a través de una prueba de concepto de un Honeypot con las características definidas en la sección anterior. Por último, en Conclusiones y trabajo a futuro se presentan posibles líneas de investigación que permiten extender el proyecto, y conclusiones finales.

[1] Se puede ver más detalle acerca de este concepto en `https://www.synopsys.com/glossary/what-is-ethical-hacking.html`

# 2. Revisión de antecedentes

En el documento *Estado del Arte* [1] se estudió la literatura disponible en donde se presentan diferentes definiciones sobre los Honeypots. Eric Cole, Ronald Krutz y James Conley [3] los definen como *sistemas diseñados para parecerse a algo que un intruso pueda atacar, construidos con el principal propósito de desviar los ataques y aprender de estos sin comprometer la seguridad de la red.* Por su lado, Joseph Migga Kizza [4] los define como *un mecanismo de señuelo monitoreado que se utiliza para mantener a un hacker alejado de recursos valiosos de la red y proporcionar una indicación temprana de un ataque.* Mientras, Lance Spitzner [5] define a estos dispositivos como *recursos de seguridad cuyo valor radica en ser sondeados, atacados o comprometidos.* En el contexto de este proyecto definiremos Honeypot como:

*Un dispositivo de seguridad que funciona como "cebo" llamativo para los atacantes, lo cual provoca que se mantengan alejados de los activos principales y que se pueda generar información sobre sus ataques*

## 2.1. Clasificación de Honeypots

Joseph Migga Kizza [4] y Lance Spitzner [5] clasifican a los Honeypots de dos maneras. La primera de ellas es respecto a su *propósito*, siendo estas categorías las de **Honeypots de producción**, que son aquellos presentes en ambientes de producción de una organización donde su finalidad es desviar la atención de los atacantes sobre los activos principales, y **Honeypots de investigación**, que tienen la finalidad de ser utilizados para generar información acerca de los ataques realizados sobre el mismo. Se espera que los Honeypots de producción sirvan para **prevenir ataques** (mantener alejados a los atacantes), **detectar ataques** (reconocer y alertar actividad sospechosa) y **responder ante ataques** (devolver al atacante un resultado esperado por él).

La segunda clasificación se basa en la *interacción que tiene un atacante con el dispositivo.* Ante mayor interacción, aumenta la cantidad de acciones que podrá realizar un atacante, sin embargo, también aumenta el riesgo asociado a su uso. El primer nivel dentro de esta clasificación es el de **baja interacción**, identificándose principalmente por ser un componente sencillo capaz de brindar respuestas simples a un atacante. Un nivel de **alta interacción** pretende simular completamente un servicio, sistema operativo u otro componente funcionalmente completo, acarreando consigo un alto costo de puesta en producción. Por último, el nivel de **media interacción** es una combinación de los dos anteriores, logrando un Honeypot con gran nivel de interacción a bajo costo de despliegue.

## 2.2. Principios de Honeypots

Las siguientes tres principios se infieren de los distintos papers analizados en el documento *Estado del Arte* [1]: **no interferencia**, que puede entenderse como la *capacidad de ser agregados dentro de una infraestructura sin alterar el funcionamiento previo.* **No exposición**, el cual hace referencia a que *un Honeypot no debe ser expuesto* de manera tal que reduce las posibilidades de que un usuario "normal" pueda llegar a él. Por último el principio **no establecer comunicación con usuarios legítimos**, el cual refuerza el principio anterior y refiere a *evitar la mayor cantidad de falsos positivos*, entendiendo que para acceder al Honeypot es necesario que se realice alguna actividad que pueda considerarse sospechosa.

## 2.3. Aplicaciones Web

En la actualidad muchos negocios operan mediante sitios o aplicaciones web, por lo que es de interés encontrar maneras de proteger estos puntos de entrada a una red privada. Su dificultad de protección radica en que no solo presentan un único punto de riesgo, sino que pueden potencialmente presentar varias debilidades. Respecto a esto, la organización **OWASP** (**O**pen **W**eb **A**pplication **S**ecurity **P**roject) presenta su proyecto **OWASP Top Ten Project** que recolecta las diez amenazas más comunes respecto a aplicaciones web. Para entrar en detalle al respecto se recomienda la sección de Aplicaciones Web del *Estado del Arte* [1].

## 2.4. Pasaje a Honeypots de Aplicaciones Web

Con el objetivo de reforzar la seguridad en aplicaciones web, a comienzos de la segunda década de este siglo se empieza a investigar como adaptar los conceptos de un Honeypot a nivel de red para el campo de las aplicaciones web. En este nuevo contexto es deseable que un Honeypot cumpla con tres propiedades [6]:

- **Functionality:** Las funcionalidades de una aplicación no deben verse alteradas.
- **Performance:** Cambios en la arquitectura del sistema no pueden implicar perjuicios a los usuarios.
- **Security:** El Honeypot no puede presentar vulnerabilidades a explotar que permitan al atacante controlar la aplicación; a su vez se debe restringir el acceso desde el Honeypot a la aplicación.

En la actualidad, existe una variedad de implementaciones de Honeypots de aplicaciones web, disponible para diferentes lenguajes de programación. A continuación, se muestran métricas respecto al mantenimiento general de las implementaciones estudiadas y los lenguajes utilizados para el desarrollo:


Figura 1: Lenguajes de desarrollo

En Figura 1 se identifica que tanto Python como PHP lideran las elecciones de cada desarrollador para la creación de sus Honeypots seguido por Java, y existe un conjunto mínimo donde se destaca la presencia de una implementación con NodeJs, un lenguaje sumamente usado hoy en día.


Figura 2: Mantenimiento de implementaciones

En la Figura 2 se puede comprobar que existe una gran carencia en el mantenimiento de las herramientas, ya que más de la mitad de los Honeypot mencionados no posee actualizaciones en el último año. Este punto es de suma importancia si se considera como una reacción desalentadora

al uso de Honeypots de aplicación web. De todas formas, varios de esos casos son demasiado acotados a una realidad específica, e incluso en uno de ellos el creador aclaró que por falta de tiempo no podía realizar el mantenimiento necesario.

## 2.5. Content Management System

Muchos de los sitios y aplicaciones web actuales son desarrollados empleando alguna herramienta de manejo de contenido, más conocidas como **CMS:** [7] **C**ontent **M**anagement **S**ystem, siendo Drupal [8], Joomla! [9] y Wordpress [10] algunos de los más conocidos y utilizados.

Estos CMS permiten agregar características a sus sitios mediante *plugins*, donde algunos de estos presentan el comportamiento de un Honeypot como *Honeypot Toolkit* un ejemplo para *Wordpress* o *drupo* ejemplo para *Drupal* vistos en la sección dedicada a CMS en *Estado del Arte* [1]. Todos ellos pueden, a alto nivel, ser modelados de una de dos formas: mediante la adición de un campo oculto en un formulario, o mediante un objeto de tipo captcha [2]. En el contexto de los Honeypots propiamente dichos, estos "Honeypots de CMS" pueden denominarse **Honeytokens**.

## 2.6. Honeytokens y Breadcrumbs

Un *Honeytoken* puede ser entendido como un recurso estático ubicado en sistemas o redes con el propósito de atraer la atención de ciber criminales y poder obtener información sobre las vulnerabilidades de sus sistemas. El autor *Spitzner* define en **The Other Honeypot** [11] a los Honeytokens como entidades digitales que también son considerados Honeypots, donde *su valor radica en su abuso y no en su uso*. Esto implica que estos elementos deben ser alterados para que se pueda inferir información en base a ellos. Además un, Honeytoken permite a los administradores monitorear sus sistemas y redes, como también obtener información sobre los atacantes que logran burlar las medidas de seguridad dispuestas.

Existen varios tipos de Honeytokens, como son: direcciones de email inválidas que se despliegan en el servidor de correo para que ante su uso se evidencie una intrusión, datos inválidos en una base de datos atractivos para el atacante, archivos ejecutables inválidos que recolectan información y la envían al propietario del ejecutable una vez el atacante accede al archivo, entre otros.

También se conocen varias implementaciones de estos artefactos como honeybits [12] que en servidores de producción despliegan información ficticia como historial de ejecución de comandos o historial de navegación. Otros ejemplos de Honeytokens, en este caso desarrollos a medida, son honey*ku* [13] en base a Heroku [14] para creación y monitoreo de endpoints HTTP falsos y DCEPT [15] destinado a su uso en el Active Directory de Microsoft. Se destacan los reconocidos

[2]Son pruebas desafío-respuesta controladas por máquina que son utilizadas para determinar cuándo el usuario es un humano o un programa automático

canarytokens[3] que representan todo un proyecto destinado a ser un servidor que permite crear Honeytokens y monitorear la red.

Los Honeytokens pueden agruparse formando **breadcrumbs**, los cuales tienen como finalidad encaminar a un atacante hacia un Honeypot. En este contexto, es necesario que los Honeytokens se vean como información real para el atacante, logrando que estos recolecten dicha información que conforma el breadcrumb, haciéndolos creer que la misma es valiosa para la organización.

Entre los ejemplos de breadcrumbs se tienen accesos ficticios a cuentas de usuarios para simular actividad en ellas, o documentos y mails ficticios que se descargaron en un directorio compartido.

## 2.7. Sesión

En el contexto de investigación abordado en este informe, se refiere a **sesión** como un *mecanismo para reconocer todas las acciones realizadas por un único atacante*. Estos mecanismos, por ejemplo, son de gran ayuda al momento de clasificar si un conjunto de ataques se asocian a una herramienta o a un atacante y a partir de ello se puede reconocer particularidades de los ataques generados por cada uno.

Dentro de la sección *Sesiones de Usuarios* del documento *Estado del Arte* [1] se describen algunos de los mecanismos utilizados para darle seguimiento a los usuarios en el contexto de las Aplicaciones Web. Uno de los métodos más comunes es el uso de las **Cookies**, las cuales brindan un *manejo de estado en la comunicación entre servidor y cliente*, con lo cual es posible que un usuario realice diferentes transacciones conservando cierta información entre ellas. Las cookies pueden ser clasificadas según el origen de la comunicación, donde se encuentran las **first-party cookies** (asociadas al sitio que se visita) y **third-party cookies** (utilizadas por sitios de terceros que proveen contenido al sitio que se visita); o clasificadas según el criterio de almacenamiento, donde se encuentran las **non-persistent cookies** (activas mientras se esté utilizando el navegador) y **persistent cookies** (almacenadas por el navegador una vez que la sesión finaliza).

Posterior a las cookies surgen las **SuperCookies**, las cuales implementan mejoras sobre las funcionalidades de las cookies con las cuales se facilita el seguimiento de usuarios. Algunos ejemplos de este tipo de cookies son las *flash cookies*, *zoombie cookies* y *permaCookies*, estos tres casos brindan un mecanismo diferente al convencional para el almacenamiento de los valores de sesión asociados a los usuarios que se conectan a las aplicaciones web.

Además del uso de cookies, las aplicaciones web presentan otras componentes que brindan la posibilidad de realizar un manejo de sesión las cuales son: **Variables de Sesión en HTML5** almacenadas en la variable global *window*, llamadas *SessionStorage* y *LocalStorage*. **Cached**

[3]Permiten implantar trampas en sus sistemas de producción en lugar de instalar Honeypots separados, para mas detalle ver la subsección Canarytokens del documento de Estado del Arte.

**Data Fingerprintig** que consiste en el uso de los cabezales HTTP *Last-Modified* e *ETags*, los que suelen carecer de control sobre el valor que emiten. Por último, **Browser Fingerprinting** es otro mecanismo donde se realiza seguimiento de los usuarios a partir de la información generada por ellos mismos desde sus navegadores. Para este último caso mencionado existen dos aplicaciones llamadas *AmIUniq?* y *PanoptiClick* que analizan esta información brindada por los navegadores para poder darle al usuario un valor acerca de que tan fácil puede ser reconocerlo.

Un aspecto importante a destacar es que *todos estos mecanismos analizados dentro del Estado del Arte son utilizados para darle seguimiento a los usuarios.* Esto quiere decir que **los atacantes podrían llegar a evitar estos mecanismos**, dado que se considera que obtener un seguimiento de las acciones de un atacante es un problema muy complejo que no será abordado en su totalidad dentro de este trabajo.

# 3. Análisis y Diseño

En el comienzo de esta sección se presenta el análisis realizado para interiorizarse en los conceptos más importantes sobre Honeypot para aplicaciones web, para ello se incorporan definiciones, se establecen las componentes y se analizan herramientas necesarias para desplegar este tipo de Honeypots. Además se crea el concepto de *Honeypot Embebido* y se incurre en el uso de Honeytokens en el contexto del proyecto.

Posterior al análisis, se continua con el trabajo realizado en la etapa de diseño. Esta etapa se compone de la construcción y especificación de las últimas características necesarias para dar lugar a la comprensión total del sistema. Mediante el uso de diversos diagramas se entra en detalle sobre como se comunican los distintos componentes y que responsabilidades tiene cada uno.

## 3.1. Objetivos

El objetivo a nivel de proyecto es mediante la investigación exhaustiva, alcanzar la especificación de un *Honeypot que pueda ser utilizado para detectar posibles intrusiones en aplicaciones web de producción, sin interrumpir en la operativa con los usuarios de dichas aplicaciones.* A su vez considerando que el comportamiento del mismo debe permitir extensiones y cubrir una amplia variedad de aplicaciones, se establece que será un framework [4] que cumpla con los siguientes objetivos:

1. **Extensible** El diseño del framework debe permitir la adición de nuevos componentes que permitan implementar tanto nuevas funcionalidades como disponer de mejores versiones o alternativas a las ya existentes.

2. **Genérico** El framework debe ser lo suficientemente genérico para ser utilizado por diferentes aplicaciones web.

3. **Obtención de información** Debe contar con capacidad para recolectar información útil sobre ataques y/o atacantes.

4. **Detección de intrusos** Debe proveer un mecanismo para discernir entre usuarios legítimos y maliciosos.

5. **Seguimiento de atacantes** Es deseable que cuente con algún mecanismo que permitan dar seguimiento al proceder de cada atacante.

6. **Sin interrupción operativa de la aplicación web** Es necesario que no se interrumpa en la operativa habitual de la aplicación web, ya que su presencia debe pasar desapercibida.

## 3.2. Análisis de HTTP y HTML

Las aplicaciones web están compuestas de diferentes tecnologías, tanto en componentes de tipo hardware como software, pero en el estudio abordado en este proyecto se hace énfasis en dos componentes: **protocolo HTTP** y **lenguaje HTML**.

### 3.2.1. Protocolo HTTP

El protocolo HTTP es el que permite *recibir* todos los elementos de una aplicación web (mediante mensajes denominados *response*) para ser mostrada por un navegador, y *enviar* las peticiones (mediante mensajes denominados *request*) realizadas desde navegador hacia el servidor. Más allá del funcionamiento, es interesante analizar la anatomía de los mensajes, los cuales

[4] Se llama *framework* [16] al esquema o estructura que se establece y que se aprovecha para desarrollar y organizar un software determinado

pueden ser divido en tres secciones [17]: primera línea (cuyo contenido varía según si el mensaje es un request o un response), los encabezados (o *headers*) y el cuerpo (o *body*), siendo éste opcional. El nombre de cada elemento puede verse en la siguiente figura:

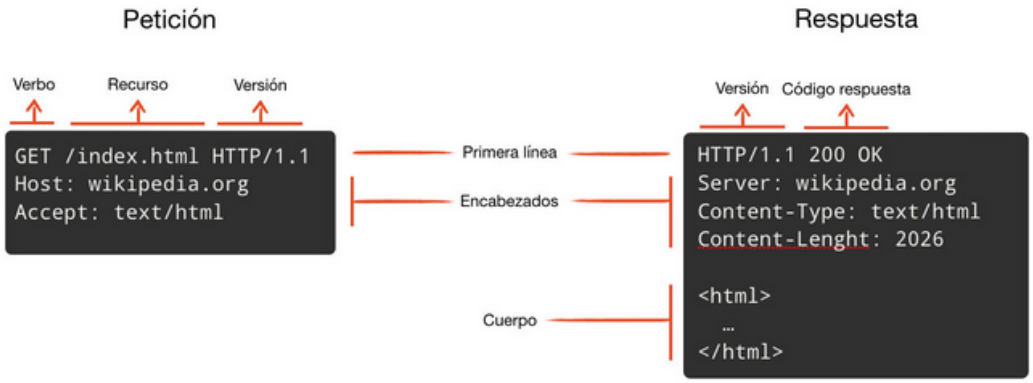


Figura 3: Anatomía de un mensaje HTTP

En el contexto del proyecto se define que las aplicaciones a las cuales se hará referencia serán aquellas basadas en el protocolo HTTP, y las secciones de interés son aquellas que transportan metadatos y contenido de la aplicación.

- **Headers** Es la sección que provee información o metadatos sobre el mensaje HTTP en forma de parámetros con su respectivo valor (único o lista de valores), subdivido en cuatro categorías pues se tienen headers generales, para request y response.
- **Body** Es la sección donde esta el cuerpo (si lo hay) de un mensaje HTTP, por lo tanto se utiliza para transportar dicho contenido.

### 3.2.2. Lenguaje HTML

HyperText Markup Language (HTML por sus siglas en inglés) es un lenguaje que permite la visualización de aplicaciones web (junto con tecnologías como CSS y JavaScript) basado en el uso de *etiquetas*, las cuales permiten la definición de distintos elementos y propiedades de la interfaz visual de la aplicación. Por ejemplo, la etiqueta *form* permite la definición de un *formulario* en una aplicación web.

Otro ejemplo destacado son los inputs que son elementos HTML que pueden contener y transportar datos entre un usuario y una aplicación web. Muchos usuarios malintencionados suelen emplear herramientas automáticas que escanean el código HTML de estas aplicaciones e intentan explotar cada elemento, ejecutando todos los enlaces que encuentran en su camino, así como inyectar información en todos los inputs a los que accede. Como es una herramienta, puede llegar a no distinguir entre campos ocultos o visibles.

## 3.3. Honeypot a desarrollar

Un camino posible para lograr cumplir con el objetivo principal propuesto en Objetivos es enfocarse en un desarrollo de *Honeypot Embebido en Aplicaciones Web* basado en *Honeytokens*, los cuales estarán *dentro* del activo principal y serán los utilizados al momento de definir si las acciones de un usuario pueden ser catalogadas como sospechosas o no.

Para poder comprender lo mencionado en el párrafo anterior se considera importante citar, la definición de **Honeypot Embebido de Aplicaciones Web**, instanciada en el documento *Análisis y diseño* [2], donde se define a este tipo de sistemas como:

*"Es un caso particular de Honeypot de Aplicación Web donde se busca incrementar la atracción del atacante y dirigir sus ataques al Honeypot desde el mismo activo dado que simula ser parte de él."*

El camino para llegar a esta definición es en base a la definición de **Honeypot de Aplicaciones Web** (también instanciada en el mismo documento), la cual a su vez surge de la definición de **Honeypots** presente en la sección Revisión de antecedentes.

Un aspecto importante a destacar acerca de los **Honeypots Embebidos** (sin discriminar el contexto de trabajo en el que se definan) es que en lo relacionado a los **Tipos de Honeypots**, se considera apropiado clasificarlos como **Honeypots de Baja Interacción**, debido al alto riesgo que se tiene por desplegarse formando parte de la misma aplicación. Los **Honeypots de Baja Interacción** en el contexto de las Aplicaciones Web también se encuentran definidos en el documento *Análisis y diseño* [2].

En Honeytokens y Breadcrumbs se introduce el concepto de Honeytokens y cuál es su importancia, donde la principal aclaración es que se entenderá que "abuso" es sinónimo de "manipulación" , pues la simple interacción con un Honeytoken será suficiente para evidenciar que se está en presencia de un usuario malintencionado. Es necesario definir que por *manipulación* se entiende a la *acción de modificar el estado en que se presenta un Honeytoken al usuario del sistema*. Es decir que si un usuario altera el contenido de un Honeytoken (lo que se definirá más adelante en Representación de Honeytokens en el Honeypot para cada tipo de Honeytoken), así como si elimina un Honeytoken, será suficiente para catalogar su interacción como malintencionada. Como se puede apreciar, esta interacción con los Honeytokens es catalogada como **Baja Interacción**, dado que solamente se modifican o eliminan valores de recursos estáticos, los cuales no llegan a ser considerados por la Aplicación Web.

Con el uso e inclusión de los Honeytokens *dentro* de la aplicación web, se asegura que el Honeypot solo procesará dichos artefactos, teniendo en cuenta la información que estos contienen para poder cumplir con los objetivos propuestos. *Cualquier otra información de la aplicación web no será procesada ni analizada por el Honeypot.*

Se considera apropiado aclarar que el término **Honeypot** refiere al framework a desarrollar. A su vez, el mismo es considerado un **Honeypot Embebido** debido a que los **Honeytokens** son inyectados por dicho framework sobre componentes de la Aplicación Web.

El Honeypot a desarrollar debe ser una instancia particular de la máquina de estados presentada en el documento *Análisis y Diseño* [2]. Dado que el objetivo principal de la herramienta es poder detectar la intrusión de un usuario y registrar información relevante al mismo, se desarrolla la siguiente instancia *reducida*:

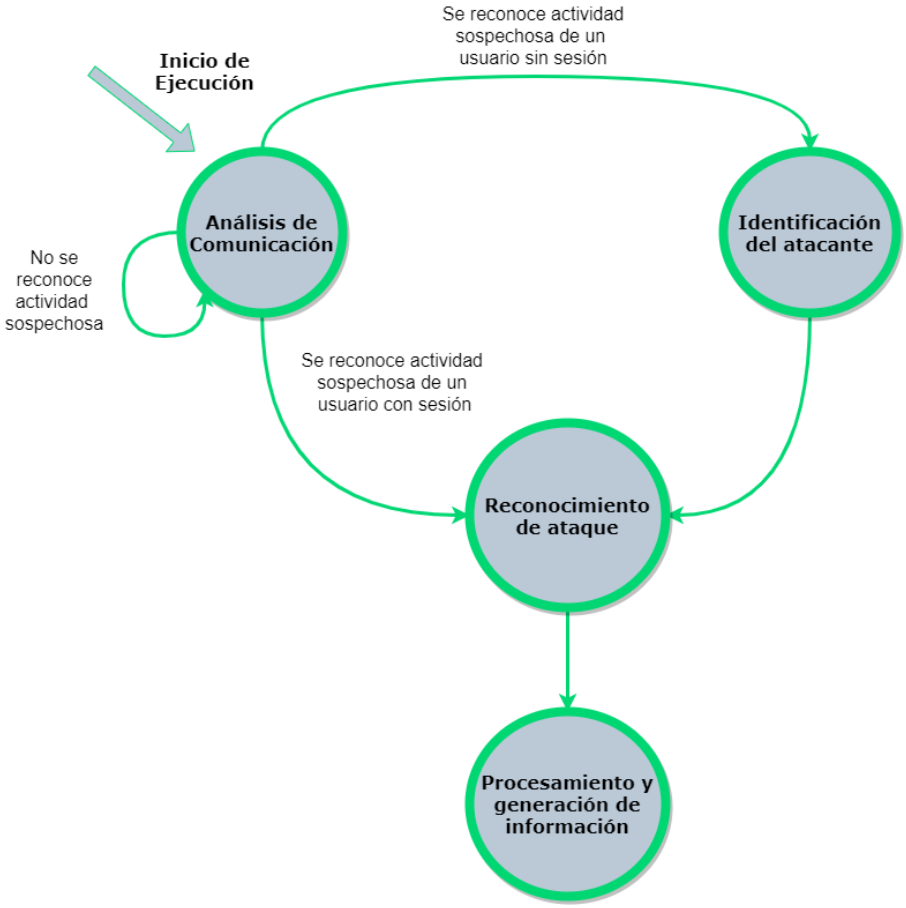


Figura 4: Máquina de estados del Honeypot Embebido a desarrollar

A continuación, se describe el comportamiento de los estados y las funcionalidades relacionadas a cada uno de ellos:

- **Análisis de comunicación**

  En este estado se analizan los pedidos (request) en busca de indicios de *comportamiento sospechoso*.

  Por *comportamiento sospechoso* se entiende a la *manipulación de Honeytokens* tal como fue presentada con anterioridad.

En este estado se definen las siguientes funcionalidades:

- **Análisis de mensajes HTTP**

  Debido que el marco de estudio comprende a las Aplicaciones Web, es necesario tener la capacidad de analizar y manipular mensajes del **Protocolo HTTP**, el cual es utilizado para la comunicación en este tipo de aplicaciones.

- **Detección de comportamiento sospechoso**

  Se entiende que un mensaje contiene "comportamiento sospechoso" si alguno de los Honeytokens allí presentes ha sido manipulado.

- **Identificación de un usuario malintencionado**

  Cuando el sistema recibe mensajes de parte de un usuario, se debe tener la capacidad de diferenciar si dicho usuario ha sido identificado como malintencionado anteriormente.

- **Identificación del atacante**

  Este estado representa una etapa en donde se *asigna un identificador único para reconocer al usuario ya clasificado previamente como malintencionado.*

  En este estado se define la funcionalidad:

  - **Generación de sesión para el usuario**

    Se deberá contar con un mecanismo de asignación y manejo de sesiones para los diferentes usuarios que ya fueron reconocidos previamente como atacantes.

- **Reconocimiento del ataque**

  El Honeypot tendrá como finalidad detectar si se está en presencia de un usuario malicioso mediante el análisis de los Honeytokens presentes en el request. Este objetivo no representa que se esté necesariamente frente a un ataque, por lo que, si bien el estado se representa en la máquina de estados, no se reconocerán los ataques perpetuados por un usuario malicioso.

- **Procesamiento y generación de información**

  La información recabada por el Honeypot debe ser disponibilizada para que otros puedan acceder a ella utilizando algún mecanismo para normalizarla y compartirla con otros técnicos especializados o dispositivos de seguridad.

  En este estado se definen las siguientes funcionalidades:

  - **Normalización de información**

    Es deseable que la información procesada en este estado se lleve a un formato estandarizado y de esta forma pueda ser consumida por mas de un sistema externo al cual se deseé disponibilizar.

- **Registro de información**

  Se espera registrar todos los datos emitidos por el atacante mediante sus mensajes HTTP, los mensajes de respuesta de la Aplicación Web y la información que el Honeypot genere relacionada a los mensajes catalogados como alarmantes.

- **Envío de información a un servidor remoto**

  Se espera tener la capacidad de enviar la información generada a un servidor remoto en donde sea más seguro mantenerla en caso de que el servidor donde se encuentra el Honeypot sea comprometido.

## 3.4. Componentes del Sistema

El siguiente diagrama es una representación de alto nivel del despliegue del Honeypot:

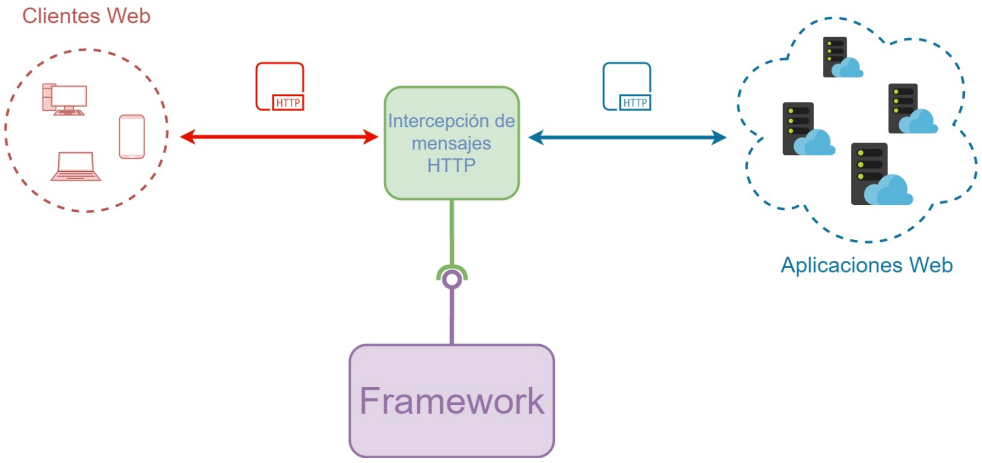


Figura 5: Diagrama de componentes del Honeypot

Como se observa, **Aplicaciones web** es el conjunto de aplicaciones que ofrecen servicios que consumirán los distintos **Clientes web** a demanda. Algunos de estos clientes web buscan *explotar vulnerabilidades* sobre estas aplicaciones que les permitan obtener datos confidenciales o incluso tomar control de la infraestructura. De esta manera se incorporará un **Interceptor de mensajes HTTP** en la comunicación entre cliente y aplicación. Este *interceptor* puede ser cualquier elemento que pueda interponerse en la comunicación entre cliente y servidor (un proxy, algún tipo de *plugin* en caso de tratarse de un CMS, o incluso la aplicación misma), y estará encargado de redirigir los mensajes hacia el **Framework**.

Será responsabilidad del Honeypot el manejo de los Honeytokens presentes en el mensaje que disponibiliza el *interceptor*. **El Honeypot no deberá leer, obtener o eliminar de ninguna manera ningún tipo de información que no esté contenida y/o sea parte de un Honeytoken.**

El siguiente diagrama ilustra el comportamiento del Honeypot:

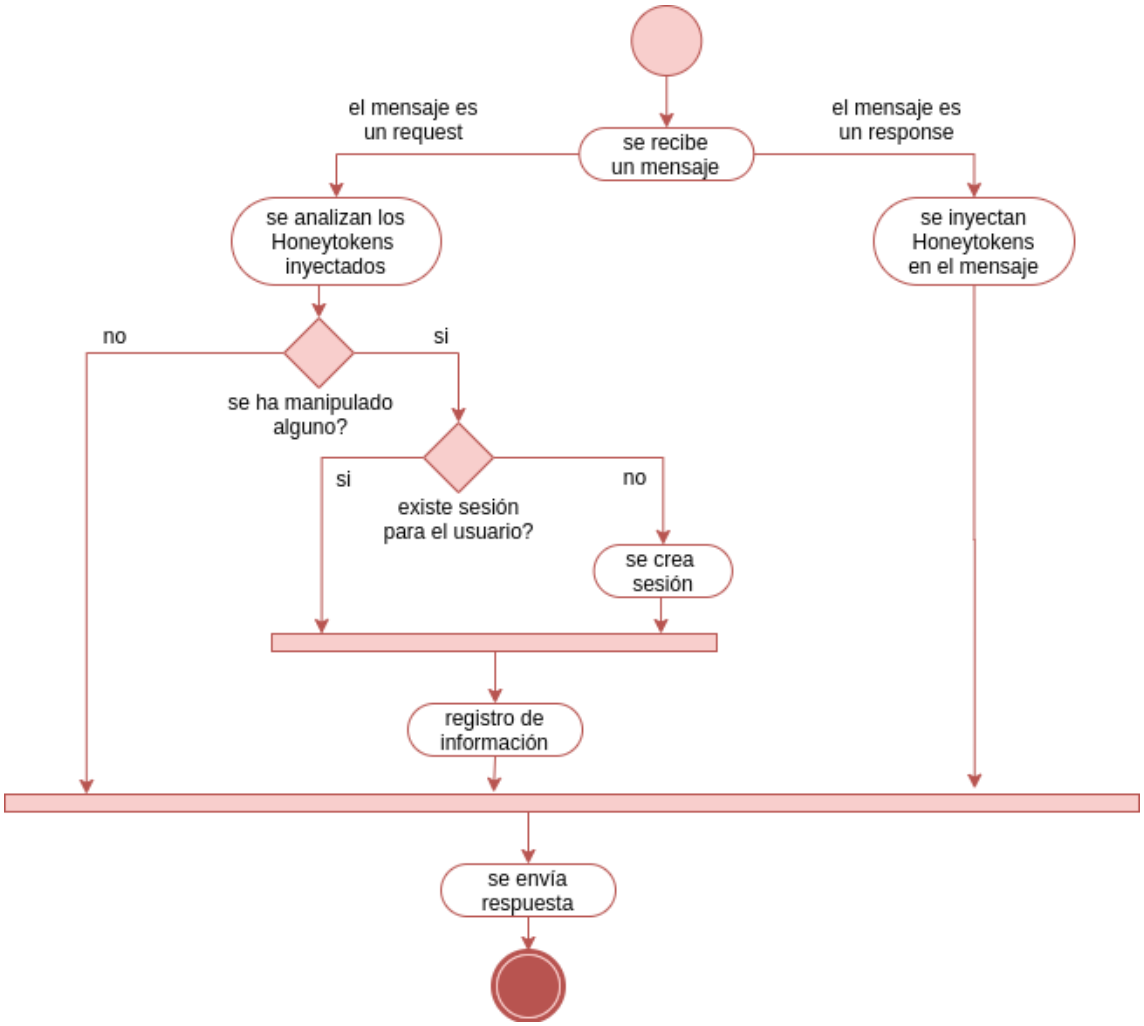


Figura 6: Diagrama de flujo del Honeypot

La ejecución del flujo comienza al recibir un mensaje HTTP. Estos mensajes deberán ser catalogados en *mensajes emitidos por la aplicación web (response)* o *mensajes emitidos por un cliente (request)*.

Los mensajes response son los mensajes a los que se deberán *inyectar* Honeytokens. Los mensajes request recibidos serán analizados para determinar si los Honeytokens inyectados en el response correspondiente han sido manipulados. En caso de que se detecte una manipulación se deberá dar seguimiento al usuario responsable del request procesado. Es decir, se le deberá crear una *sesión* a ese usuario (si es que no tenía una previamente). Además, se almacenará información con el objetivo de tratar de identificar al usuario y su accionar catalogado como intrusivo.

## 3.5. Interacción con el Honeypot

Para que el sistema comience su funcionamiento, es necesario que exista una entidad o componente encargado de entregar al Honeypot los mensajes emitidos por la aplicación web y los clientes. El siguiente diagrama de secuencia ilustra esa comunicación:

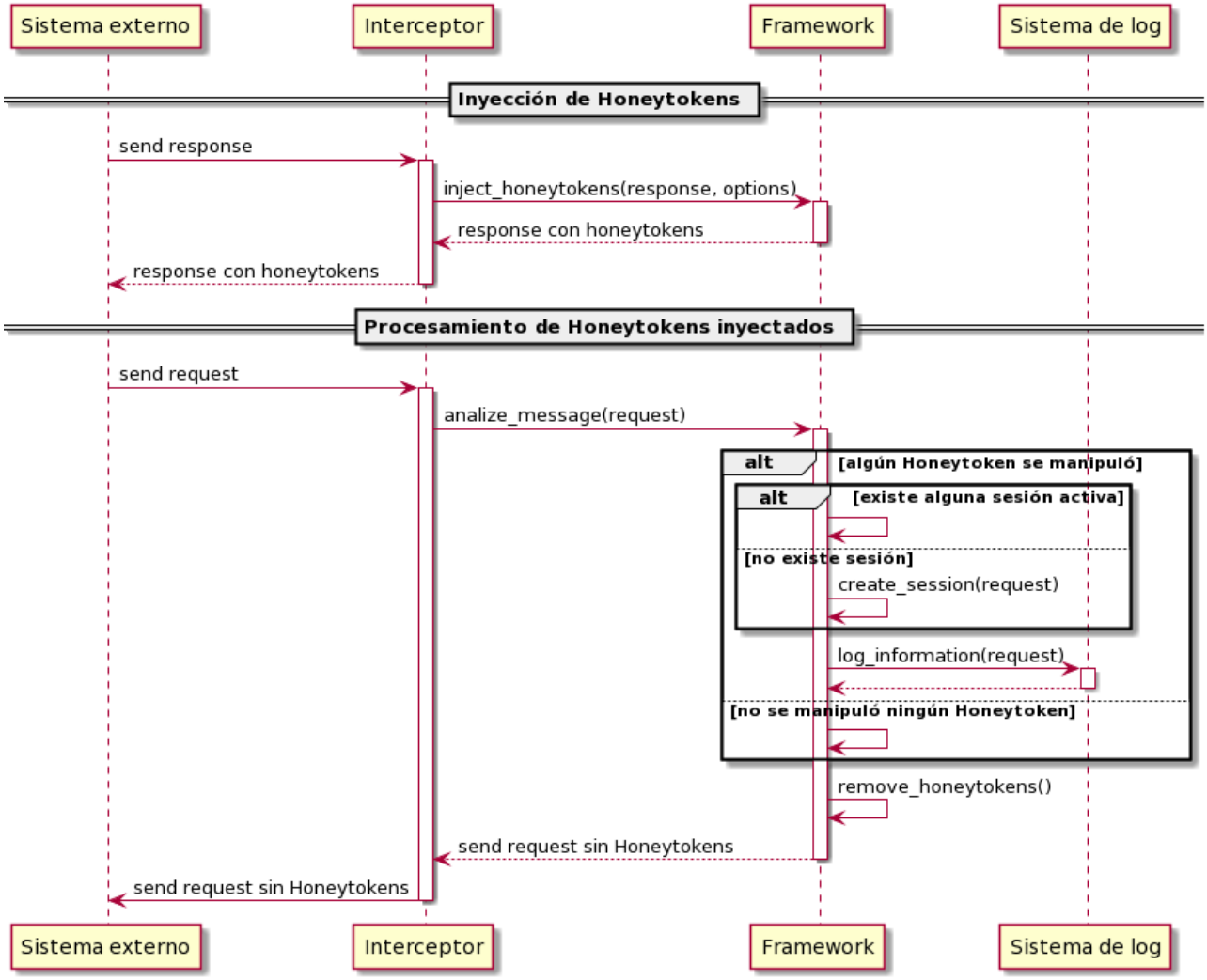


Figura 7: Llamadas al Honeypot

La entidad **sistema externo** hace referencia al emisor de los correspondientes mensajes (para el caso de un request sería un usuario, y para el response la aplicación web), mientras que **interceptor** hace referencia a esa entidad capaz de dirigir los mensajes hacia el **framework**. Se deduce que esta entidad debe ser capaz de ubicarse *en medio* de la comunicación entre cliente y aplicación web.

Para el caso en que se precise inyectar Honeytokens, se dispondrá de la operación *inject_honeytokens*, la cual recibirá el mensaje response. En el caso del análisis de un request, se dispondrá de la operación *analize_message* que recibirá el mensaje request a procesar. Antes

de que el framework retorne el mensaje a la entidad intermediaria, los Honeytokens deberán ser eliminados de manera que la aplicación web procese los mensajes sin estar alterados por el Honeypot.

El diagrama anterior establece claramente las responsabilidades de cada entidad involucrada en el funcionamiento del Honeypot. La entidad intermediaria será responsable únicamente de funcionar como *pasaje* entre los mensajes y el **framework**, este último estará a cargo de saber qué Honeytokens se han inyectado en que response, además de cuál es el request asociado a ese response. Así como el manejo de sesiones que también estará a cargo del **framework**.

## 3.6. Diagrama de Despliegue

Se emplea un **diagrama de despliegue**[18] como presentación de la disposición física en alto nivel de la solución junto a sus diferentes componentes:

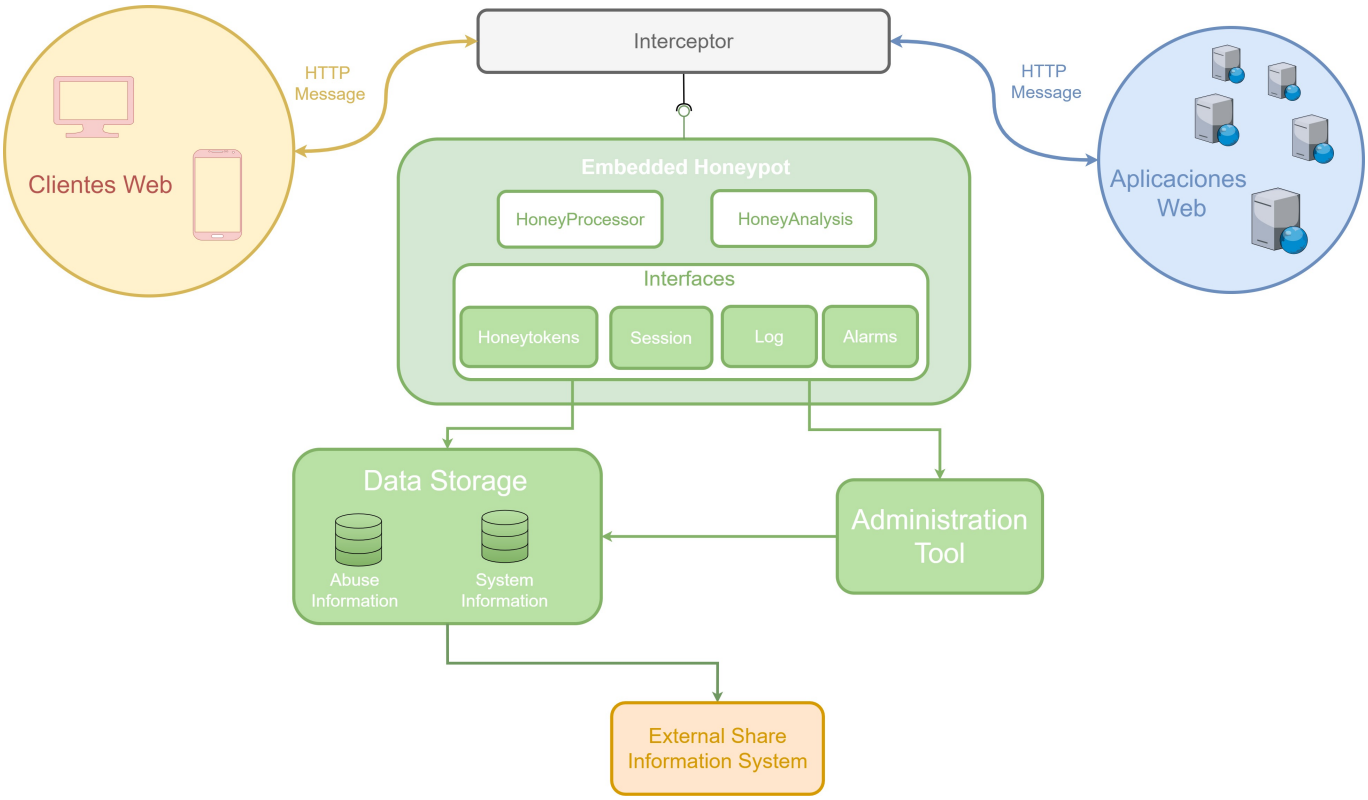


Figura 8: Diagrama de despliegue del framework

Este diagrama se compone de las siguientes partes:

### Clientes Web

Los *Clientes Web* representan a *todos los usuarios que pueden conectarse a las Aplicaciones Web*. Con "*todos los usuarios*" se hace referencia tanto a usuarios normales como malintencionados.

Los clientes podrán conectarse a las aplicaciones utilizando una conexión mediante un navegador web en cualquier dispositivo que lo permita (PC, dispositivo móvil, etc).

### Aplicaciones Web

Las *Aplicaciones Web* hacen referencia al *sistema final al que el usuario desea conectarse*, es decir, a la aplicación en la que el framework agregará los Honeytokens para luego analizar el abuso sobre ellos.

**Interceptor**

El Interceptor es la componente encargada de *interceptar los mensajes HTTP emitidos entre los clientes y las Aplicaciones Web para luego ser utilizados por* ***Embedded Honeypot*** *para su posterior modificación y análisis*. Básicamente, esta componente es utilizada para ponerse cómo intermediario de la comunicación y consumir del framework para hacer el análisis de los mensajes. Como se menciona anteriormente, esta componente puede ser cualquier tipo de software que capture los mensajes y pueda comunicarse con el framework, como por ejemplo un proxy, un *plugin* o la aplicación web misma.

**Framework**

El Framework puede ser dividido en tres grandes componentes, las cuales a su vez se subdividen en otras subcomponentes:

1. **Embedded Honeypot**

   *Embedded Honeypot* es la componente fundamental del framework, dado que se encarga de *procesar los mensajes HTTP que se envían entre los Clientes Web y las Aplicaciones Web para poder detectar actividad sospechosa y agregar la información necesaria en base a la actividad sospechosa detectada*. Estos mensajes, *son recibidos* desde el **Interceptor** a través de una interfaz.

   Las subcomponentes en las que se divide el mismo son:

   1.1 **HoneyProcessor**

   Esta subcomponente es la que se encarga de *agregar Honeytokens y los valores de sesión en cada mensaje* HTTP response recibido por el **Interceptor**.

   1.2 **HoneyAnalysis**

   Esta subcomponente se encarga de *definir si un request HTTP emitido por un cliente contiene un abuso*. En caso de detectar un abuso, la misma componente deberá *crear un identificador de sesión para ser utilizado para darle seguimiento al usuario malicioso* y también *almacenar la información acerca del comportamiento del usuario y dar un aviso a los administradores, respectivamente.*

   1.3 **Interfaces**

   Esta subcomponente contiene las cuatro interfaces principales del framework: **Honeytokens**, **Session**, **Log** y **Alarms**. Estas interfaces son las que definen las operaciones necesarias para realizar la operativa de **HoneyProcessor** y **HoneyAnalysis**.

   Cabe destacar que al presentarse como interfaces, estas componentes permiten al framework ser extensible, lo cual ayuda que se pueda tener diferentes implementaciones de las operaciones definidas para cada una. En la sección Vista de operaciones se ilustra con mayor profundidad las operaciones involucradas en dichas interfaces.

2. **Administration Tool**

   La Herramienta de Gestión es la componente *encargada de brindarle una interfaz a los administradores para que puedan comunicarse con el framework y así realizar las configuraciones necesarias sobre el mismo.* Estas configuraciones involucran información para el funcionamiento de todas las entidades (Honeytokens, Sessions, Alarms y Log) y valores requeridos para que el framework funcione apropiadamente (como las rutas o conexiones a bases de datos para obtener los datos de las entidades).

3. **Data Storage**

   La componente de *Data Storage* alberga la información necesaria para el funcionamiento del framework. Básicamente, contiene un repositorio encargado de mantener los datos del framework y sus entidades y otro con los datos de los ataques detectados. Esta componente implementa mecanismos para comunicarse con *External Share Information System.*

**External Share Information System**

La componente *External Share Information System* esta asociada a sistemas externos al framework a construir, los cuales brindan la posibilidad de compartir información generada por *Embedded Honeypot* hacia otros sistemas.

## 3.7. Diagrama de Clases

En la siguiente imagen se presenta un diagrama de clases reducido del framework[5]

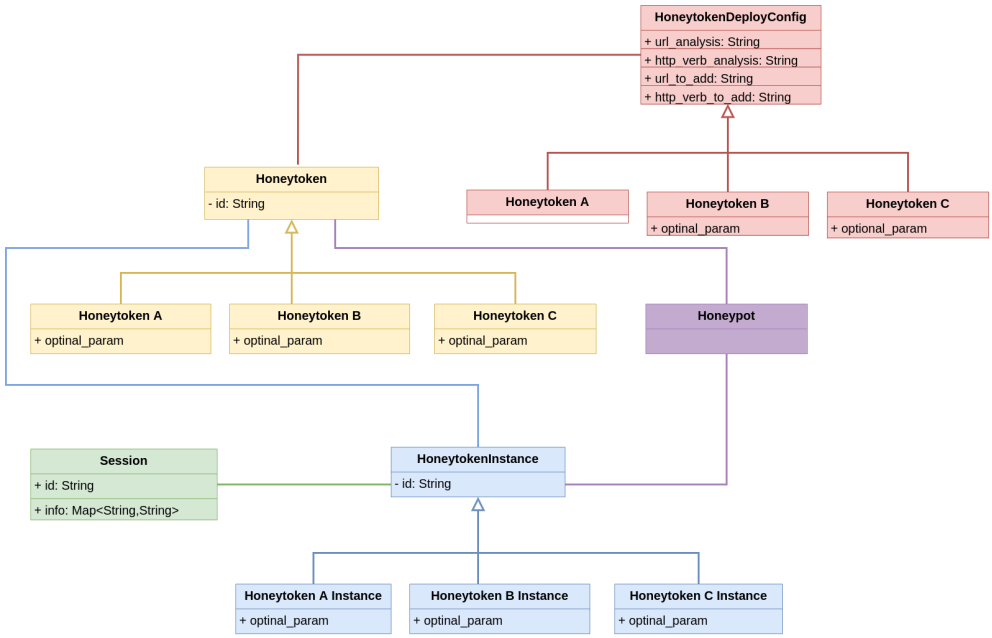


Figura 9: Diagrama de clases del framework

En este diagrama se pueden ver las entidades más importantes del framework: **Honeytoken**, **HoneytokenInstance**, **Session**, **HoneytokenDeployConfig** y **Honeypot**.

Se puede apreciar la entidad **Honeytoken**, la cuál representa los *datos estáticos definidos para los Honeytokens*, es decir, muestra la entidad que almacena los datos que brinda el administrador de la aplicación al momento de definir los Honeytokens a utilizarse. Esta entidad se define como una jerarquía, *con lo cual se brinda un mecanismo para que los diferentes desarrolladores que utilicen el framework puedan crear distintos tipos de Honeytokens*, brindado de esta forma la *posibilidad de extender esta arquitectura*. Es considerable aclarar que estos **Honeytoken** contienen atributos comunes a todos los tipos de Honeytokens que se puedan definir, pero cabe destacar que es posible definir atributos particulares para cada tipo de Honeytoken.

Los **HoneytokenInstance** persiguen el objetivo de *establecer el concepto de instancia de Honeytokens definidos para el framework*, es decir, *mostrar una entidad que tiene la responsabilidad de presentarse dentro de los mensajes HTTP intercambiados entre los clientes y las aplicaciones web, que a su vez es una instancia de un Honeytoken previamente definido en el framework por los administradores de la aplicación*. Esta idea ayuda a separar a la entidad presente dentro de las interacciones entre el cliente y las aplicaciones web con entidad que se define a partir de los datos brindados por los administradores de la aplicación web. También se puede apreciar que

[5]Se considera que el diagrama es reducido dado que el mismo no presenta los manejadores, controladores y datatypes involucrados, solamente se basa en las entidades necesarias para el funcionamiento.

esta entidad se presenta como una jerarquía, esto se debe a que está asociada directamente a la entidad **Honeytoken**, por lo que es necesario crear por cada tipo de **Honeytoken** con un tipo de **HoneytokenInstance**.

En el momento en que se introducen los **HoneytokenInstance** se presenta el problema de *identificar un Honeytoken dentro de un mensaje enviado por un cliente* y *reconocer en cuál mensaje debe ser enviado el Honeytoken*, para lo cual se crea la entidad **HoneytokenDeployConfig**. Esta entidad contiene los datos necesarios para que dado un mensaje HTTP se reconozca que Honeytokens analizar o agregar en el mismo. Nuevamente, es importante remarcar que esta entidad requiere de una jerarquía. Esto es debido a que cada tipo de **Honeytoken** puede requerir algún dato particular (además del dato de la URL y el verbo HTTP que se encuentran presentes para todos en el diagrama) para ser analizado o agregado a los mensajes intercambiados.

**Session** es una entidad particular, utilizada para *identificar usuarios maliciosos*. Es decir, una vez que se reconoce un abuso sobre un **HoneytokenInstance**, se *crea una Sesion para darle seguimiento a las acciones del usuario que generó el abuso*. Cada **Session** contiene un identificador único para el framework y de esta forma se asocia la información de la manipulación de uno o varios **Honeytoken** con una **Session**.

**Honeypot** es la entidad principal que funciona como puerta de entrada a la ejecución del sistema. Esta entidad es la encargada de exponer los métodos que serán invocados para agregar Honeytokens en un response y detectar la manipulación en alguno de ellos.

## 3.8. Vista de operaciones

En el capítulo Diagrama de Clases se establece una visión sobre la información contenida dentro de cada entidad del framework. Una vez que se logra establecer los atributos de cada entidad, es necesario conocer cuáles son sus responsabilidades. Es decir, *cuáles son las operaciones que cada entidad ejecutará*.

A continuación, se muestra un diagrama que contiene algunas de las componentes del framework ya vistas en Diagrama de Despliegue que serán de utilidad para presentar las operaciones mas relevantes del framework:

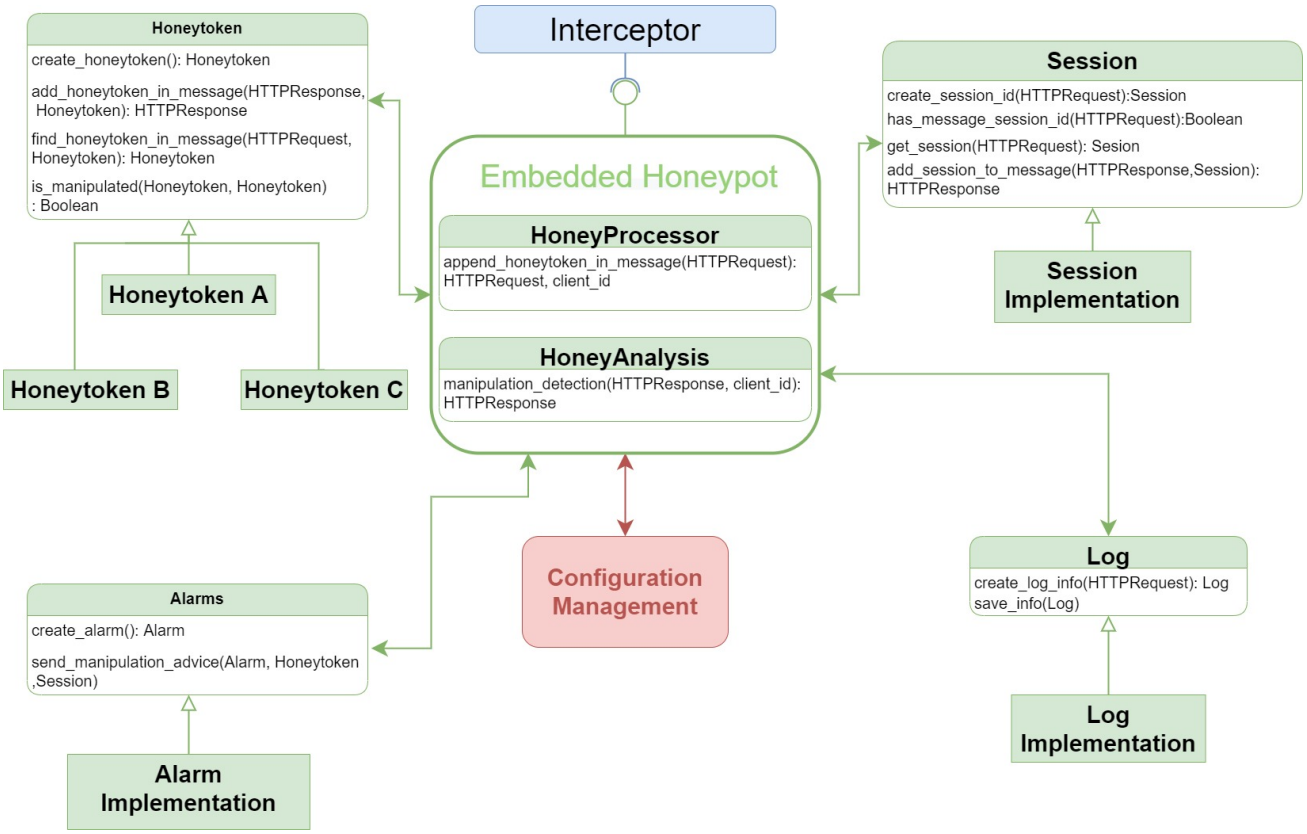


Figura 10: Diagrama con operaciones del framework

En el diagrama se puede apreciar nuevamente que el **Interceptor** se conecta a una interfaz habilitada por **Embedded Honeypot**. En dicha interfaz se encuentran las dos operaciones principales del desarrollo:

- **append_honeytoken_in_message**: Esta operación toma un mensaje HTTP response y le agrega los HoneytokenInstance que estén asociados a la URL de donde proviene la solicitud a la cual se le está respondiendo.
- **manipulation_detection**: Esta operación recibe un mensaje HTTP request y detecta si los *HoneytokenInstance* asociados a la URL dentro del mensaje han sido manipulados.

Las componentes **HTTP Processor** y **HTTP Analysis**, los cuales implementan las operaciones anteriormente mencionadas, requieren de otras componentes del framework para poder realizar la operativa asociada a cada operación. **HTTP Processor** se comunica con **Honeytoken** y **Session** para poder obtener información de cada una de estas entidades y así agregar los *HoneytokenInstance* a los mensajes HTTP response. A su vez, la componente **HTTP Analysis** requiere de las componentes **Honeytoken**, **Session**, **Alarms** y **Log**, para reconocer, almacenar información y dar aviso a los administradores al momento de detectar un abuso sobre un *HoneytokenInstance*.

Las componentes **Honeytoken**, **Session**, **Alarms** y **Log** son presentadas como **Interfaces**, es decir, *se requiere de una nueva componente que implemente su definición*. De esta forma, se permiten agregar nuevos mecanismos para realizar cualquiera de las operaciones que se definan en estas clases, logrando que los diferentes desarrolladores que se quieran involucrar a futuro en el proyecto puedan agregar nuevos tipos de Honeytokens, nuevas formas de aviso a los administradores, mecanismos de sesión o métodos de registro de información con facilidad.

Por otro lado, se presenta una comunicación con una componente llamada **Configuration Management**. La finalidad de dicha componente en este caso es disponer la información relacionada a la configuración general del framework y los datos de las entidades (incluyendo los datos de la clase *HoneytokenDeployConfig* presentada anteriormente).

# 4. Implementación

El análisis y diseño planteados corresponden a una especificación que busca ser genérica y extensible, como se establece en Objetivos, dado que los Honeytokens del Honeypot son fácilmente intercambiados mediante la implementación de una interfaz.

De manera de validar la investigación realizada, así como saber que elementos deben ser ajustados para una correcta definición del sistema, se decide realizar una *prueba de concepto* (*POC*, **P**roof **O**f **C**oncept) sobre un Honeypot embebido dentro de un CMS.

A lo largo de esta sección se brinda en detalle como serán implementadas las componentes del sistema definidas en la etapa de diseño, y particularidades de los Honeytokens a utilizar.

## 4.1. Representación de Honeytokens en el Honeypot

En esta sección se describirán los Honeytokens a utilizarse a partir del análisis de su ubicación dentro de protocolo HTTP y cómo será su forma de detección e inyección en los mensajes.

### 4.1.1. Consideraciones sobre Honeytokens

Antes de definir los tipos de Honeytokens a utilizarse dentro del framework se considera importante destacar dos particularidades relacionadas con esta entidad dentro del sistema:

1. A nivel operacional, cada tipo de Honeytoken debe implementar la interfaz de Honeyotoken que se define en Vista de operaciones. Por lo tanto, cada tipo de Honeytoken implementa su propia forma de ser desplegado y la funcionalidad que confirma su manipulación.

2. Respecto a los datos dentro de los tipos de Honeytokens, es considerable aclarar que cada uno cuenta con la capacidad de definir atributos particulares a si mismo, respetando la definición de atributos generales a todos estos, como se aprecia en Diagrama de Clases.

Es importante destacar que en la sección Objetivos se establece que el Honeypot no puede alterar el funcionamiento de la herramienta a la que se adhiere. En base a esta premisa se considera que cada tipo de Honeytoken debe cumplir:

*Los Honeytokens deben ser capaces de ser agregados dentro de la infraestructura de la organización sin alterar el funcionamiento previo.*

### 4.1.2. Tipos de Honeytokens

Como es abordado en la sección Análisis de HTTP y HTML, los mensajes HTTP se pueden dividir en tres secciones: la primera línea, el conjunto de headers y el body. Para la definición de los tipos de Honeytokens a utilizarse, el interés está en las últimas dos secciones, pues tienen

la capacidad de ser extendidas, brindado la posibilidad de inyectar los Honeytokens dentro de ellas.

A continuación se describen los tipos de Honeytokens que se definen para la implementación, considerando su forma de agregarse a los mensajes HTTP, la forma de detectar la manipulación y los datos necesarios de cada uno:

1. **Header generales**

   Un header HTTP está compuesto por un par "nombre:valor", donde el protocolo no define restricciones respecto a cómo debe estar formado el nombre, ni en la cantidad de headers que pueden coexistir pero es necesario que el nombre sea único, ya que en caso contrario puede ocurrir problemas con sus respectivos valores. Esta libertad en el manejo de los headers HTTP es lo que permite que las aplicaciones puedan definir los suyos propios con mayor especificidad.

   A partir de esta observación se define la siguiente restricción sobre este tipo de Honeytokens, la cual debe ser cumplida por los administradores que configuren la herramienta:

   *El atributo **nombre** del Honeytoken no puede coincidir con ninguno existente en la aplicación web.*

   Estos Honeytokens deberán ser inyectados en algún punto del ciclo de comunicación con el usuario. Por lo expuesto al momento, tiene sentido pensar que el Honeypot inyectará un header en el response que se envía al usuario, pero el protocolo HTTP no tiene manera de asegurar que un header enviado en un response sea enviado automáticamente por el cliente en un request posterior. Es por esto que **un Honeytoken de este tipo debe contener un componente que sea empleado desde el cliente, que permita inyectar los headers recibidos en los subsecuentes requests**.

   Si bien el Honeytoken será inyectado por parte del cliente, es necesario que el par *nombre:valor* sean conocidos por el Honeypot. Es por eso que **el sistema no sólo deberá inyectar en los mensajes *response* la rutina para agregar el Honeytoken en el siguiente *request*, sino que además los valores de *nombre:valor* que tendrá el Honeytoken**.

   Dado que los sitios emplean JavaScript, el Honeypot deberá inyectar en los response correspondientes una rutina en este lenguaje que permita la adición del header a los requests siguientes. En base a [19] y [20] se entiende que la rutina JavaScript deberá hacer uso de los *Service Workers* provistos por el lenguaje. Básicamente, un service worker permite

interceptar los requests con destino al servidor luego que son enviados por el cliente. En la intercepción es cuando se deberá inyectar el Honeytoken.

Respecto al abuso de los headers generales, se entenderá que existe abuso si al procesar un request no se encuentra el cabezal o si el valor recibido no coincide con el esperado.

2. **Cookies de navegación**

   Las *cookies de navegación* fueron ampliamente analizadas en el documento *Estado del Arte* [1]. Podrían incluirse en el tipo previamente descrito, pero se enfatiza en ellos porque pueden contener otros parámetros asociados como son la fecha de expiración o el uso de flags como HttpOnly o Secure, lo que las diferencia de cualquier otro cabezal.

   Sobre las cookies también se aplica la restricción de *Header generales* respecto a que no se debe colisionar con el uso de cookies de la aplicación web. De igual forma el abuso de una cookie corresponde al de un *Header general*, donde se consideran la eliminación de la cookie o la modificación en el valor de la misma. Cabe destacar que a diferencia de los *Header general*, las cookies son almacenadas por los clientes web una vez que estos las reciben.

3. **Inputs ocultos**

   Un input es un elemento HTML capaz de contener y transportar datos entre un usuario y una aplicación web. Muchos usuarios malintencionados suelen emplear herramientas automáticas que escanean el código HTML de un sitio e intentan explotar cada elemento, ejecutando todos los enlaces que encuentran en su camino, así como inyectar información en todos los inputs a los que accede. Como es una herramienta, puede llegar a no distinguir entre campos ocultos o visibles.

   Los inputs definen varios atributos, siendo *id* el más importante de todos. Es deseable que el id sea único para identificar un elemento de manera unívoca. Pero, aunque establezca un id único en un Honeytoken de este tipo, es necesario recordar que los sitios ejecutan además código JavaScript el cual contiene funciones que permiten obtener un elemento por cualquiera de sus atributos.

   Dado que los inputs se ubican *dentro* del sitio web al que acceden los usuarios, deben ser dispuestos de manera que no sean manipulados por cualquier usuario sino solo por aquellos malintencionados. Además como se estableció en Consideraciones sobre Honeytokens, *los Honeytokens no pueden alterar el funcionamiento previo de un sitio*. Por estas razones es que los inputs deben estar *ocultos* dentro del sitio en que se inyectan.

   El atributo básico de un input es el *type* y dependiendo de su valor, el elemento se comportará de distinta manera. En [21] se definen todos los posibles tipos de inputs que existen,

de los cuales *button*, *image* y *reset* **no serán tenidos en cuenta**, dado que estos inputs definen componentes visuales y no pueden ser empleados para enviar información de vuelta hacia la aplicación web.

Para que el valor ingresado por un usuario en un input llegue hacia la aplicación web es necesario que el input sea agregado como parte de un elemento *form* de HTML. Además, es necesario definir el atributo *name* del input a inyectar. Por último, el valor de los inputs que se encuentren dentro de un elemento form serán enviados hacia la aplicación web siempre y cuando el mismo no sea vacío. Es por esto que **será necesario que cada Honeytoken inyectado de este tipo contenga un valor por defecto** de manera que pueda evaluarse correctamente si el mismo fue modificado o no.

En este punto se puede formular la restricción para *Inputs ocultos*:

*Es necesario que estos Honeytokens definan una cantidad mínima de atributos, siendo estos el **type**, **id**, **name** y **value**. Este último contendrá un valor no vacío, y **name** e **id** deben tener valores únicos respecto al sitio en que se inyectan.*

Existen varias condiciones que definen el *abuso* de uno de estos Honeytokens. Si se cumple al menos una de ellas se entenderá que un Honeytoken ha sido modificado:

- Si se altera el atributo *name*
- Si se altera el valor por defecto que contiene el Honeytoken por otro no vacío
- Si se elimina el valor por defecto del Honeytoken
- Si se elimina el atributo *value* del Honeytoken

## 4.2. Alcance

Es importante remarcar cuáles serán los límites del desarrollo en base al trabajo de análisis y diseño realizado hasta el momento. Por lo tanto, a continuación serán instanciados algunos de las de los diagramas realizados.

### 4.2.1. Diagrama de Despliegue

Para comenzar a delimitar el alcance del desarrollo se comienza con el diagrama de despliegue realizado en Diagrama de Despliegue. Su instancia para el desarrollo queda definida en el siguiente diagrama:

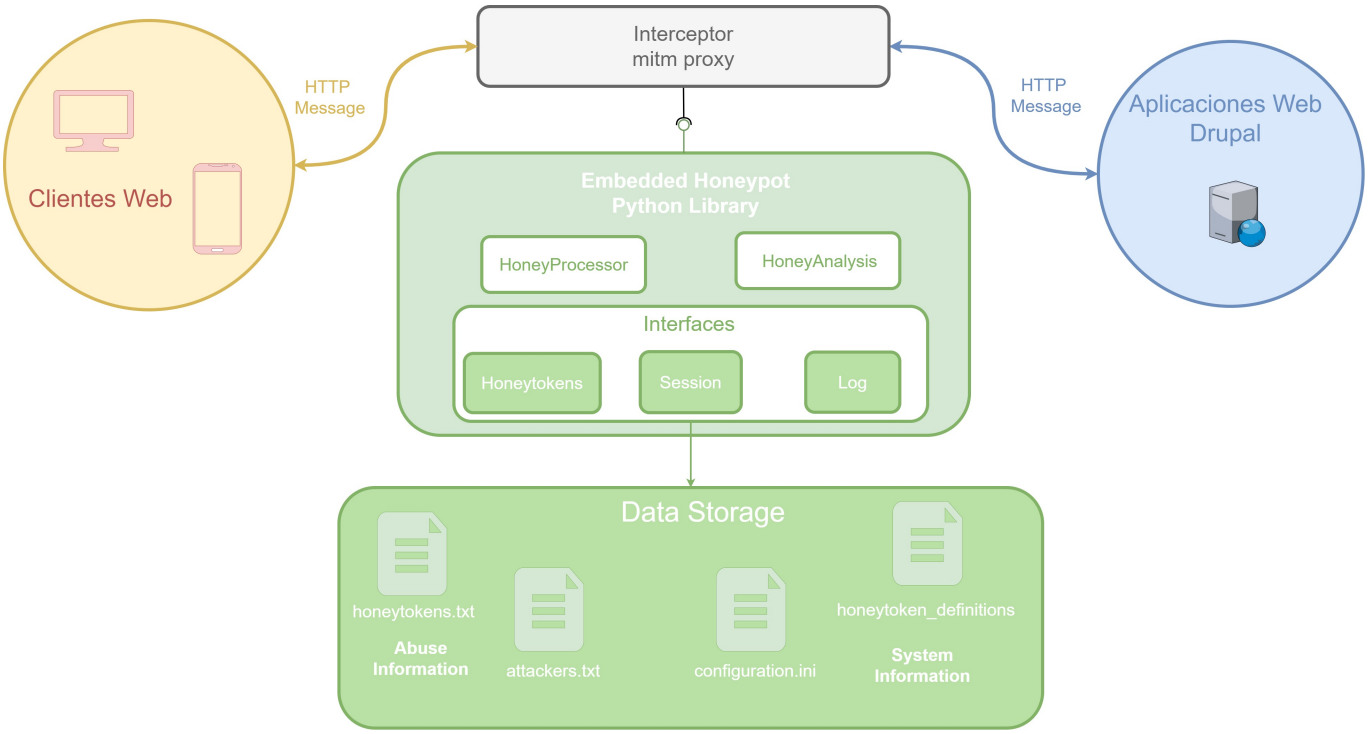


Figura 11: Diagrama de despliegue instanciado para el desarrollo

Como se aprecia, en esta instancia se tiene el mismo conjunto de clientes web mencionados anteriormente pero que en este caso se conectan a una aplicación **Drupal**.

Se decide por el uso de la herramienta **mitm proxy** para que implemente la componente de **Interceptor**. Este proxy es un desarrollo en Python el cual brinda un flexible manejo sobre los mensajes HTTP capturados. Para su uso en la infraestructura el mismo es configurado como un *Proxy Reverso*.

Para la implementación de la componente **Embedded Honeypot** se decide por el uso del lenguaje Python. Esta decisión es tomada en base a que se tiene una gran cantidad de desarrollos de Honeypots de Aplicaciones Web en este lenguaje, sumado a la gran cantidad de bibliotecas

que existen, las cuales pueden facilitar parte del desarrollo, reduciendo de esta forma el tiempo invertido en la implementación.

Una de las primeras diferencias a destacar con el diagrama original es que no se encuentra la interfaz de **Alarms**. Esta componente decide no implementarse debido a que se opta por dar un mayor foco al procesamiento de los Honeytokens. Debido a esto, solamente quedan implementadas las interfaces **Honeytoken**, **Session** y **Log**, las cuales serán descritas en mayor profundidad dentro de Diagrama de Clases y Diagrama de Operaciones, donde se ilustra con más detalle la extensibilidad del framework.

Por otro lado, se puede ver que la componente **Administration Tool** tampoco se encuentra en el diagrama. Esta componente se decide no implementar debido a que para el manejo de datos de entidades del framework se utilizan archivos de texto como se puede apreciar en la componente **Data Storage**.

Dentro de la información del sistema (**System Information**) en **Data Storage**, se tiene los archivos *configuration.ini*, con configuraciones generales del framework para su funcionamiento, y *honeytokens_definitions* con la definición de los datos de los Honeytokens a utilizarse. Por otro lado, para el almacenamiento de la información de abuso de Honeytokens (**Abuse Information**) se tiene los archivos *attackers.txt*, con información de las sesiones de los usuarios a los que se les detectó un abuso sobre algún Honeytoken, y *honeytokens.txt*, donde se registra la información de abuso sobre los Honeytokens desplegados.

La componente **External Share Information System** tampoco se encuentra en el diagrama debido a que no se realiza la implementación de un mecanismo de comunicación con un tipo de herramienta que pueda compartir información acerca de los abusos detectados.

### 4.2.2. Diagrama de Clases y Diagrama de Operaciones

Como se menciona anteriormente en Representación de Honeytokens en el Honeypot, se decide por la implementación de tres tipos de Honeytokens: **HTMLInputs**, **Cookies** y **HTTP-Headers**.

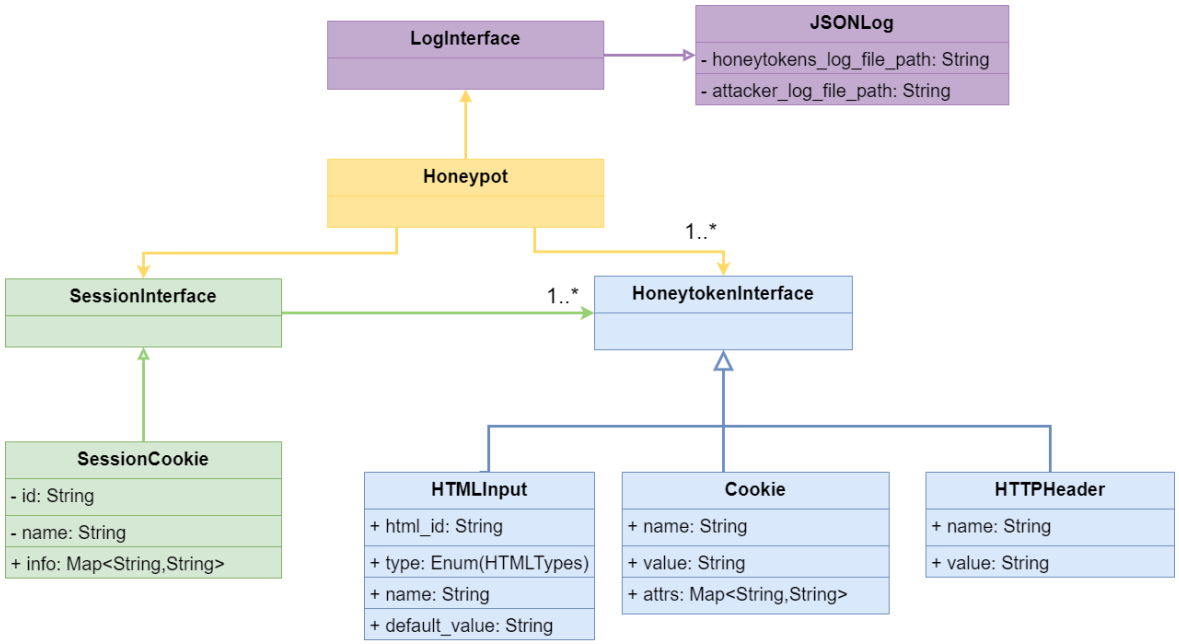


Figura 12: Diagrama de clases instanciado para el desarrollo del framework

En Figura 12 se pueden ver los atributos necesarios para los Honeytokens a implementar dentro de cada una de sus respectivas clases (**HTMLInputs**, **Cookies** y **HTTPHeaders**). A su vez, como se comenta en Diagrama de Clases, cada uno de estos tipos de Honeytokens tienen una forma particular para inyectarse y ser analizados a partir de los atributos con que son creados. La clase **Honeypot** es la encargada de ofrecer los métodos que permiten inyectar Honeytokens dentro de un response, y analizar un request para detectar posibles manipulaciones.

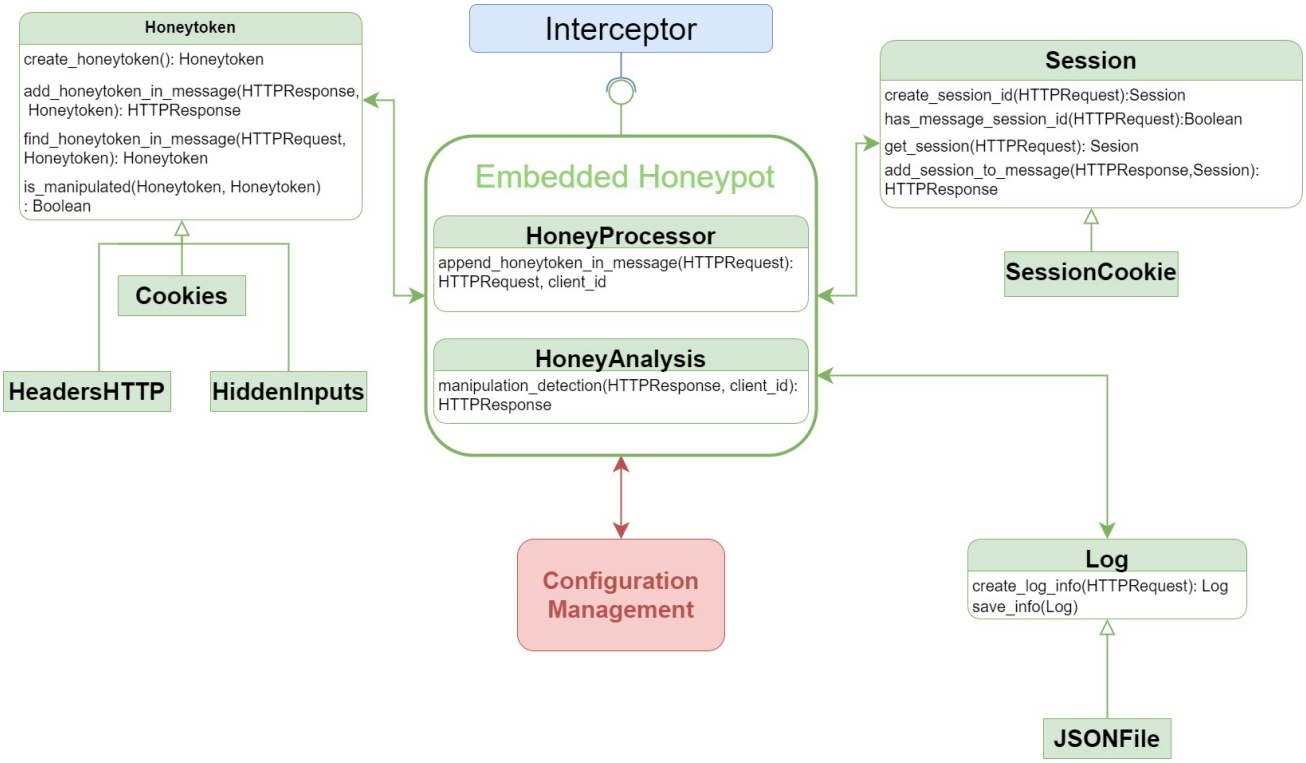


Figura 13: Diagrama con operaciones instanciado para el desarrollo del framework

En Figura 13 no se ven operaciones particulares para cada una de las clases **HTMLInputs**, **Cookies** y **HTTPHeaders**. Esto es debido a que cada una debe implementar las operaciones de la interfaz **Honeytoken**, considerando cada particularidad mencionada en Representación de Honeytokens en el Honeypot. Esto es una de las muestras de lo *extensibile* que es la herramienta, dado que se pueden agregar nuevos tipos de Honeytokens fácilmente, al igual que se incorporan *plugins*.

Lo mismo ocurre con **Log** y **Session**, dado que ambas interfaces son implementadas por **JSONFile** y **SessionCookie**, respectivamente. **JSONFile**, como se detalla en Persistencia de información, implementa la persistencia de información del sistema a partir de un archivo en formato JSON y **SessionCookie**, como se detalla en Manejo de sesiones, implementa un seguimiento de las acciones de los usuarios a través del uso de cookies HTTP. Estas interfaces también vuelve a mostrar lo extensible que es la herramienta, dado que se permite la incorporación de nuevos mecanismos para manejo de sesión y de registro de información.

## 4.3. Manejo de sesiones

En el capítulo anterior se menciona el uso de la entidad **Session** para el *reconocimiento de las acciones de los usuarios malintencionados*. Para lograrlo es necesario resolver dos problemas:

1- **Crear un identificador único** Para poder reconocer las acciones de un atacante en particular, es necesario brindar alguna forma de identificar estas acciones. Es por esto que se define el uso de un *identificador único*, el cual brinde la posibilidad de saber que atacante realizó determinado abuso sobre un Honeytoken.

2- **Recuperar el identificador en cada mensaje** Es necesario que dado un mensaje se tenga un mecanismo para recuperar el identificador asignado al atacante y de esta forma, en caso de detectar un nuevo abuso sobre un Honeytoken, se pueda agregar dicho identificador en el registro de sesiones. Para esto, es necesario definir cómo el Honeypot puede encontrar dentro del mensaje este identificador, y en caso de que sea necesario, cómo puede agregarlo.

A partir de estas consideraciones, se toma la decisión de utilizar *un generador aleatorio para crear los identificadores y que estos sean intercambiado mediante cookies*, la cual contiene el valor entre cada request y response.

Se considera sumamente importante destacar que *este mecanismo puede verse como uno de los menos apropiados*, pero **se toma en encuentra su uso debido a que no es de vital importancia dentro de este proyecto el abordaje de este problema**, dado que **dar seguimiento de las acciones de los atacantes se considera un problema sumamente complejo**. Por lo tanto, este proyecto prioriza contar un mecanismos simple que pueda ser sustituido en caso de lograr el desarrollo de un mecanismo más eficiente.

## 4.4. Persistencia de información

Como ya se ha establecido con anterioridad, cada vez que el Honeypot detecte que se está en presencia de un usuario malintencionado se registrará información relacionada a ese usuario, así como su proceder para alterar un Honeytoken y sobre que Honeytoken se dio el abuso.

La información persistida se almacenará en formato JSON. Para los datos del usuario malintencionado (los cuales se almacenan dentro del archivo *attacker.txt*), se tiene el siguiente formato:

```
{
    "ipAddress" : IP en formato IPv4
    "userAgent" : user agent del usuario
    "date"      : fecha y hora en que se detecta el acceso del usuario
    "sessionId" : identificador de sesión asignado al usuario
}
```

Es de interés almacenar también cómo fueron manipulados los Honeytokens por este usuario. Estos datos (los cuales se almacenan dentro del archivo *honeytokens.txt*) se relacionarán con los anteriores respecto al atributo *sessionId*, y respeta el siguiente formato:

```
{
    "sessionId"         : identificador de sesión asignado al usuario
    "honeytokenType"    : tipo del Honeytoken
    "honeytokenInfo"    : información general del Honeytoken
    "date"              : fecha y hora en que se registra el abuso
    "abuseDescription"  : breve descripción del abuso perpetuado
    "abuseInfo"         : datos recabados del Honeytoken abusado
}
```

En la anterior especificación, *honeytokenType* puede contener uno de los valores *HTML_HEADER, COOKIE* o *INPUT*. Por su parte, *honeytokenInfo* contiene la información perteneciente al Honeytoken: si es de tipo Header contendrá el nombre y su valor, si es Cookie contendrá nombre, valor y los atributos seteados, y para Input ocultos contendrá todos sus atributos con sus respectivos valores (type e id, entre otros).

## 4.5. Infraestructura

Para la implementación del POC se utilizan tres hosts, los cuales serán implementados mediante contenedores virtuales en la herramienta *Docker*. El siguiente diagrama de red ilustra la Infraestructura establecida:

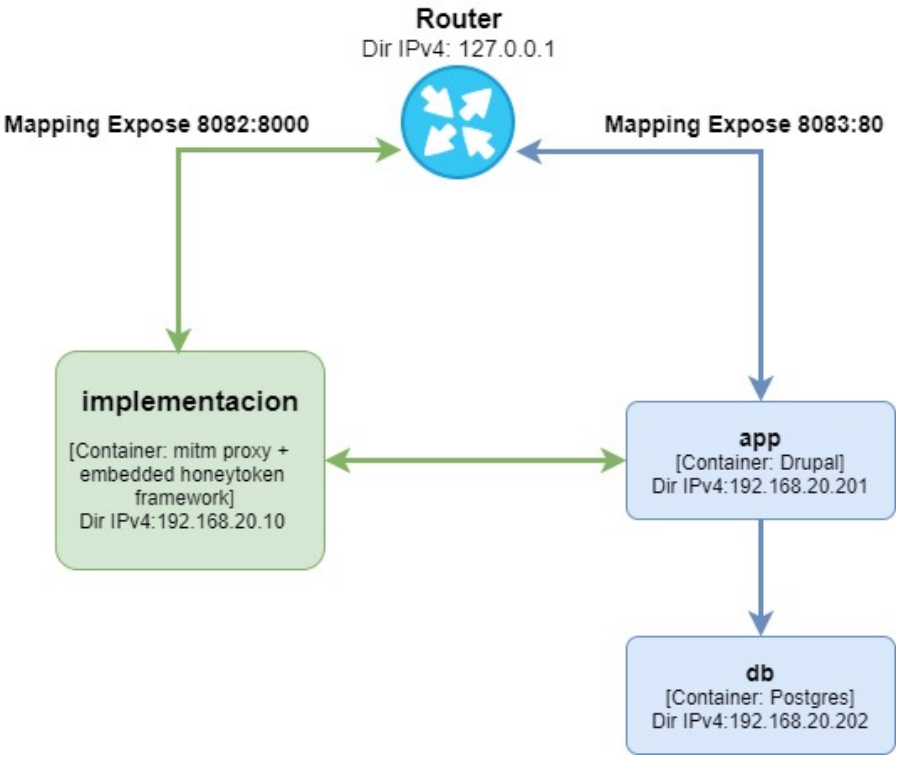


Figura 14: Diagrama de red de la prueba de concepto

Los host **app** y **db** representan la aplicación Drupal a la cual se le agregan los Honeytokens y su base de datos asociada, respectivamente. Dentro del host nombrado como **implementación**, se tiene configurado el servidor **mitm proxy** en modo *Proxy reverso* hacia el host **app**. También, dentro de **implementación**, se encuentra instalado y configurado el framework desarrollado para ser utilizado por el servidor **mitm proxy**, de manera que este tenga la capacidad de *agregar y analizar la manipulación de los Honeytokens*.

Se puede apreciar en Figura 14 que se tienen dos mapeos de puertos para realizar conexiones a los dos servidores presentes en el diagrama. Para la conexión a la aplicación web sin pasar por el servidor que contiene el framework, se establece mediante el puerto 8083. En estos mensajes no se mostrarán los Honeytokens, dado que no se establece una comunicación atravesando el Honeypot. Para acceder a la aplicación web a través del servidor, donde se agregan los Honeytokens, es a través del puerto 8082. Ambos puertos deben ser accedidos a partir de la dirección IP configurada en el Router, la cual en el diagrama se presenta como 127.0.0.1.

## 4.6. Configuración del POC

Para la configuración del componente *Implementación* mostrado en Figura 14 se emplean dos archivos, uno para la configuración global del sistema y otro que contiene la definición de los Honeytokens a emplear:

- El archivo **configuration.ini** contiene configuraciones que aplican a nivel global respecto al POC. Las propiedades que permite definir son:
  - *token_config_file_route*: contiene la ruta al archivo de log en que se almacenará la información de los tokens abusados. El formato en que se almacenan estos datos es el presentado en Persistencia de información.
  - *user_config_file_route*: contiene la ruta al archivo de log donde se almacenará la información de los atacantes detectados. El formato en que se almacenan estos datos es el presentado en Persistencia de información.
  - *cookie_name*: permite definir el nombre que tendrá la cookie de sesión empleada.
  - *definition_file*: contiene la ruta al archivo que lista las definiciones de los Honeytokens a emplear.
- El archivo **honeytoken_definitions** contiene las definiciones de los Honeytokens, y su ubicación viene dada por la propiedad *definition_file* definida antes. Cada Honeytoken se define en una nueva linea respetando una sintaxis dada, teniendo en cuenta que las líneas que comienzan en "#" *no son procesadas*.

  En general la definición es de la forma:

  honeytoken_class_name, attr_1, attr_2, ..., attr_n | injection_url, injection_http_verb, inj_prop_1, ..., inj_prop_m | detection_url, detection_http_verb, detect_prop_1, ..., detect_prop_k

  En la primera parte de la configuración se encuentran las propiedades características del Honeytoken a emplear: *honeytoken_class_name* es el nombre de la clase que representa al Honeytoken, la cual debe contener la lógica para su inyección y detección dentro de los mensajes e implementar la interfaz *HoneytokenInstance*, y *attr_1*, ..., *attr_n* son los atributos necesarios para instanciar el Honeytoken.

  La segunda parte de la configuración contiene atributos necesarios para realizar la inyección del Honeytoken. Las primeras dos propiedades, *injection_url* e *injection_http_verb*, deben ser definidas por todos los Honeytokens, mientras que las propiedades *inj_prop_1*, ..., *inj_prop_m* dependen del Honeytoken. En el caso de las dos propiedades requeridas, una o ambas pueden estar vacías (quedando el espacio vacío). En el caso de *injection_url* se

debe definir la URL de inyección del Honeytoken mediante el uso de *expresiones regulares* [22], e *injection_http_verb* puede ser cualquiera de los *verbos HTTP* [23] existentes. Una observación a este caso es que los valores *inj_prop_1*, ..., *inj_prop_m* son enviados a la clase Honeyoken como una cadena de caracteres separados por coma.

La última parte de la configuración contiene los atributos necesarios para realizar la detección de del Honeytoken configurado. Al igual que en el caso anterior, las primeras propiedades *detection_url* y *detection_http_verb* son requeridas, siendo las restantes *detect_prop_1*, ..., *detect_prop_k* atributos particulares del Honeytoken a configurar. También, al igual que antes, las propiedades requeridas pueden estar vacías: *detection_url* es la URL en la cual se detectará el Honeytoken (pudiendo definirse mediante expresiones regulares) y detection_http_verb puede ser alguno de los verbos HTTP.

Cabe destacar que si *injection_url* no se define, el Honeytoken se inyectará en todas las URL (siempre que se cumplan todas las demás condiciones). Lo mismo ocurre con *detection_url* si se define vacía, se analizarán todas las URL en busca del Honeytoken. Una situación similar ocurre cuando alguno de los verbos HTTP *injection_http_verb* o detection_http_verb se definen vacíos.

Como ejemplo de configuración, se presentan los valores permitidos en alto nivel para los Honeytokens de tipo Cookie e Input oculto.

- **Input oculto** permite la siguiente configuración:

  HTMLInputInstance,input_id,input_type,input_name,input_default_value | injection_url,injection_http_verb,form_http_method,form_id,form_action_url | detection_url,detection_http_verb

  Se observa que *HTMLInputInstance* es el nombre de la clase que representa al Honeytoken, siendo *input_id*, *input_type*, *input_name* y *input_default_value* atributos de la clase. Como atributos de inyección, *injection_url* e *injection_http_verb* son como fueron definidos antes, mientras que *form_http_method*, *form_id* y *form_action_url* definen los atributos *method*, *id* y *action* del formulario HTTP en que se inyectará el Honeytoken. Luego, los atributos de detección *detection_url* y *detection_http_verb* son como fueron definidos antes.

- **Cookie** permite la siguiente configuración:

  CookieInstance,cookie_name,cookie_value | injection_url,injection_http_verb | detection_url,detection_http_verb

  *CookieInstance* es el nombre de la clase que representa al Honeypot, siendo *cookie_name* y *cookie_value* los atributos de la clase. Los demás atributos de inyección y detección son como se definieron antes.

## 4.7. Dificultades detectadas

Durante el proceso de desarrollo se tuvieron que enfrentar algunas particularidades respecto a cada uno de los Honeytokens y en general, las cuales se detallan a continuación:

- **Honeytokens de tipo Header:**
  - Como se expresó en Representación de Honeytokens en el Honeypot, este tipo de tokens requiere de un elemento especial de JavaScript para su funcionamiento (un *service worker*) por las razones que allí mismo se expresan. Para su uso es necesario que la aplicación web sea accedida mediante HTTPS, y esto requiere que el certificado SSL utilizado por dicha aplicación sea instalado por la componente Interceptor (en este caso el servidor Proxy). Por lo tanto agregar un Honeytoken de tipo Header estaría forzando a que la componente Interceptor deba tener instalado el certificado de la Aplicación Web, lo cual llevaría a tener un despliegue más complejo del Honeypot, no siendo lo deseable ya que se espera que su instalación tenga la menor cantidad de dependencias posible.
- **Honeytokens de tipo Cookie:**
  - Respecto a este tipo de Honeytoken se detectó el problema de *reconocer la eliminación de las cookies como un abuso*. Esto se debe a dos factores: el primero es la forma en que se realizan los análisis de los mensajes, dado que el framework agrega en un HTTP response un Honeytoken para ser analizado en el HTTP request que hará el usuario sobre la misma URL; y el segundo factor, es que hasta que un usuario no se conecte a determinada URL donde se tenga configurado el Honeytoken de cookie, la misma no será enviada al usuario para que su cliente web almacene la cookie. Por lo tanto, la primera vez que los usuarios ingresen a una URL donde se tenga configurada un Honeytoken de tipo cookie, dentro del mensaje HTTP request emitido no se presentará dicha cookie, lo cual no es un abuso, dado que a dicho usuario la cookie nunca fue enviada.
- **Consideraciones generales:**
  - Al usar cookies como mecanismo de sesión o de identificación para los usuarios no maliciosos, es posible que la misma sea eliminada por un usuario malintencionado. En este caso, no es posible volver a identificar al usuario como un atacante ni recuperar el identificador previamente asignado.
  - En Representación de Honeytokens en el Honeypot se definió como abuso de los Honeytokens la alteración del nombre, pero esto no fue implementado en el POC ya que es necesario definir en que condiciones se está frente a una alteración del nombre

de un token existente y asegurarse que no se está analizando elemento propio de la aplicación. Respecto a lo desarrollado, si no se encuentra el nombre del Honeytoken, se considera que el mismo fue eliminado (esta última consideración aplica particularmente al caso de los input ocultos, debido al problema de detectar la eliminación de las cookies comentado anteriormente).

- En Persistencia de información se espera almacenar la IP al detectar a un usuario malintencionado, pero el protocolo HTTP no lo define. Dado que el Honeypot solo maneja paquetes HTTP por lo que el valor de la IP debe ser obtenido por el interceptor.

## 4.8. Conclusiones acerca de la implementación

Se puede concluir que el trabajo de implementación realizado fue satisfactorio, dado que **se obtiene una validación de la arquitectura definida a partir del desarrollo del POC**, la cual era el objetivo perseguido de esta etapa del proyecto.

Dicho POC, cubre también en gran medida los objetivos dentro de la sección Objetivos, dado que la arquitectura definida cumple con estos objetivos y la misma queda implementada y validada con este desarrollo.

A pesar de que algunas características que se consideraban enriquecedoras (como el hecho de compartir información con otros sistemas) no completamente definidas, se puede asegurar que pueden ser fácilmente agregadas en un futuro sin modificar en gran medida la parte principal de la implementación.

Más aún, se deja la posibilidad de que el framework crezca a partir de una nuevas líneas de trabajo, muchas de ellas planteadas en la sección Trabajo futuro.

# 5. Conclusiones y trabajo a futuro

## 5.1. Trabajo futuro

El proyecto está orientado a la investigación y especificación del uso de Honeypots en aplicaciones web. Muchos elementos trascienden al alcance del proyecto por lo que se consideran apropiadas para un estudio específico, quedando establecidos como posibles lineas de investigación para la continuación del proyecto. A continuación se describen estas posibles líneas que permitan extender la investigación comenzada aquí.

### 5.1.1. Manejo de sesión

Como ya se estableció con anterioridad, un punto importante del Honeypot es la posibilidad de crear sesiones y realizar un rastreo de la actividad de los usuarios catalogados como atacantes. En la prueba de concepto realizada se creó un mecanismo de sesión basado en cookies, lo que probó no ser muy fiel para mantener la sesión ya que dichas cookies pueden ser intervenidas y luego modificadas por los atacantes, perdiendo así la sesión. En consecuencia, en caso que el atacante modifique constantemente esta cookie no podrá mantenerse ninguna sesión para el mismo.

Dentro de la sección Revisión de antecedentes se realiza un estudio acerca de los mecanismos para darle seguimiento a los usuarios, aclarando que el seguimiento para los atacantes es un problema complejo, dado que para los casos nombrados existen formas para evitarlos. Este problema se considera abierto, dado que no se ha encontrado un método optimo de resolución, pero la implementación brindada deja abierta la posibilidad de modificar el mecanismo de sesión por uno mejor.

### 5.1.2. Honeytokens

Los Honeytokens son los elementos fundamentales del Honeypot Embebido especificado en este proyecto. Se presentaron ejemplos de Honeytokens a partir del análisis realizado a la estructura de los mensajes HTTP en Representación de Honeytokens en el Honeypot, pero no se establece que sea la única forma de obtener Honeytokens, ni que estos sean los únicos.

Una mejora al trabajo realizado está en desarrollar Honeytokens de mayor complejidad respecto a su anatomía como en los mecanismos definidos para la detección de abuso, tanto para los Honeytokens existentes como para los que no.

Un posible camino a seguir puede ser el uso de *breadcrumbs* como fue definido en el documento *Estado del Arte* [1], empleando *breadcrumbs de información*, por ejemplo. Un caso sería el de divulgar dentro de la aplicación web credenciales de usuario inválidas. Estas credenciales podrían entenderse como los Honeytokens, definiendo como su abuso la detección de un acceso mediante las mismas.

En Honeypot a desarrollar se establece que el Honeypot solo procesa los Honeytokens detectados. Lo planteado antes violaría esta regla, por lo que la línea de estudio debería incluir la manera de detectar el uso de estos Honeytokens sin la necesidad de procesar tráfico que no involucre estos artefactos.

### 5.1.3. Aumentar el interés del atacante

Los Honeytokens son capaces de llamar la atencion del atacante, por lo tanto se pueden utilizar de tal forma que se establezca un orden en su despliegue con el fin de ir aumentando el interés del atacante.

Un posible camino podría ser desplegar Honeytokens cada vez mas atractivos, siendo un factor de atracción cuan difícil es abusar de dicho Honeytoken. Por ejemplo, que su abuso requiera de interacción entre sistemas o aplicaciones distintas pudiendo ser uno de esos sistemas un Honeypot clásico. Otra opción a considerar podría ser el despliegue de múltiples Honeytokens según el avance del atacante, asumiendo que su interés crece en relación a la cantidad de Honeytokens desplegados.

### 5.1.4. IOC y Automatización a través de la plataforma MISP

MISP (**M**alware **I**nformation **S**haring **P**latform) [24] es una plataforma de código abierto que permite reunir, compartir, almacenar y correlacionar Indicadores de Compromiso permitiendo la detección y prevención de ataques, y es empleada por diversas organizaciones en el mundo. La información recabada por el Honeypot podría ser disponibilizada en esta plataforma, dándole un uso que trascienda la generación de métricas.

Generar los IOC y su potencial uso, junto con la comunicación al MISP, son posibles hilos de investigación a considerar.

### 5.1.5. Recolección de ítems

Un ítem define a los archivos que pueden ser cargados por un atacante en un input de tipo *file* que ofrece una aplicación web. La recolección de ítems referencia al análisis de estos archivos, de manera de no solo evidenciar un intento de intrusión por parte de un usuario, sino también incluir información valiosa que permita describir distintos tipos de ataques. Dos ejemplos claros serían la carga de scripts que ejecuten rutinas en el servidor, y la carga de código encapsulado para vulnerar bases de datos.

Los ítems pueden verse como Honeytokens complejos, como se definió más arriba. La información recolectada mediante el análisis de los archivos cargados permite aumentar las medidas de seguridad de los sistemas a proteger, creando así distintos IOC que pueden ser compartidos entre distintos sistemas. De esta manera, puede verse que la recolección de ítems está unida a varias líneas de investigación ya presentadas.

### 5.1.6. Despliegue de alarmas ante detección de intrusión

Además del registro de las intrusiones detectadas, puede ser de interés contar con un sistema de alarmas que sea activado ante la detección de un usuario malintencionado. Este componente sería un agregado al sistema base, sumando importante valor ya que con su adición se permitiría tener registros en tiempo real sobre las intrusiones. En el documento *Análisis y diseño* [2] se encuentran posibles caminos en que este nuevo componente puede ser agregado al Honeypot.

### 5.1.7. Reconocimiento de ataques

La especificación del Honeypot permite detectar los accesos de usuarios malintencionados, pero éstos pueden modificar los valores de los Honeytokens con un valor que intente provocar un ataque especifico contra la aplicación. Puede ser de interés someter estos tokens a una revisión por parte de una herramienta especializada, de manera de detectar si el valor del Honeytoken representa un tipo de ataque existente.

A partir de lo mencionado, se podrían reconocer si los Honeytokens abusados contienen ataques conocidos o algún tipo de ataque no conocido, los cuales podrían ser de gran interés estudiar para compartir con la comunidad información acerca de los mismos.

## 5.2. Conclusiones

Como ha sido mencionado, la finalidad principal de este proyecto es "obtener un Honeypot de Aplicaciones Web, el cual pueda ser agregado en ambientes de producción sin alterar la operativa de los usuarios". Se puede garantizar que dentro de este documento queda plasmado el cumplimiento de esta meta mediante la especificación de un Honeypot embebido en aplicaciones web en base a Honeytokens y el desarrollo de un POC sobre un Honeypot de estas características.

El framework desarrollado cumple con lo definido previamente para ser un dispositivo de este tipo, dado que los Honeytokens desplegados buscan ser atractivos para los atacantes, logrando que estos pierdan la atención sobre el activo principal (las Aplicaciones Web) y se genere información al respecto en el momento en que se detecta el abuso sobre dichos Honeytokens. Es importante resaltar que en el párrafo anterior y dentro de la sección Honeypot a desarrollar se menciona que el desarrollo consiste en un **Honeypot Embebido de Aplicaciones Web**, para lo cual se crea una definición dentro del proceso de análisis del proyecto y es profundizada en el documento *Análisis y diseño* [2].

Complementando la justificación anterior, dentro de la sección Objetivos se listan algunos objetivos los cuales logran ser abordados en completitud:

1. **Genérico**: Este objetivo se cumple dado que en cualquier infraestructura se puede agregar el framework desarrollado, ya sea a partir de una nueva componente de red o en la aplicación misma.

2. **Extensible**: El framework cumple con esta característica, dado que es construido para permitir agregar nuevas componentes por diferentes desarrolladores.

3. **Obtención de información**: Se definen estructuras de datos donde es almacenada la información acerca de los atacantes y los ataques detectados.

4. **Detección de intrusos**: Se define la posibilidad de detectar atacantes a partir del abuso de Honeytokens generados por el framework.

5. **Seguimiento de atacantes**: Este objetivo no fue uno de los lineamientos principales del proyecto, sin embargo se da una solución simple, la cual se conoce que no es del todo eficiente, y se brinda la posibilidad de dejar abiertas mejoras en este aspecto.

6. **Sin interrupción operativa de la aplicación web**: La arquitectura definida y desarrollada evidencia como es posible agregar un Honeypot de Aplicaciones Web, sin perjudicar a los usuarios de dichas aplicaciones.

En conclusión, se puede afirmar que el proyecto cumple con los objetivos definidos y logra obtener una especificación que se aplica en gran medida a la realidad establecida y queda comprobado a partir del desarrollo de un POC. A su vez, el proyecto brinda nuevas lineas de trabajo, las cuales pueden enriquecer y mejorar la herramienta para el uso de toda la comunidad.

# Referencias

[1] Nicolás Zeballos Federico Pernas, Agustín Sanchez. Entregable: Estado del Arte. 2020.

[2] Nicolás Zeballos Federico Pernas, Agustín Sanchez. Entregable: Análisis y Diseño. 2020.

[3] James W. Conley Eric Cole, Ronald Krutz. *Network Security Bible*. Wiley Publishing, Inc, Indianapolis, Indiana, 2005.

[4] Joseph Migga Kizza. *Computer Network Security*. Springer Science+Business Media, Inc, Chattanooga, Tennessee, 2005.

[5] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison Wesley, Boston, Massachusetts, 2002.

[6] Thorsten Holz Michael Möter, Felix Freiling and Jeanna Matthews. A generic toolkit for converting web applications into high-interaction honeypots. 2007.

[7] Wikipedia. Sistema de gestión de contenidos - wikipedia, la enciclopedia libre. `https://es.wikipedia.org/wiki/Sistema_de_gestion_de_contenidos`. [Accedido: 09-OCT-2019].

[8] Wikipedia. Drupal - wikipedia, la enciclopedia libre. `https://es.wikipedia.org/wiki/Drupal`. [Accedido: 09-OCT-2019].

[9] Wikipedia. Joomla - wikipedia, la enciclopedia libre. `https://es.wikipedia.org/wiki/Joomla`. [Accedido: 09-OCT-2019].

[10] Wikipedia. Wordpress - wikipedia, la enciclopedia libre. `https://es.wikipedia.org/wiki/WordPress`. [Accedido: 09-OCT-2019].

[11] Lance Spitzner. Honeytokens: The other honeypot. `https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=74450cf5-2f11-48c5-8d92-4687f5978988&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments`. [Accedido: 25-AGO-2020].

[12] GitHub. Github - 0x4d31/honeybits: A poc tool designed to enhance the effectiveness of your traps by spreading breadcrumbs & honeytokens across your systems to lure the attacker toward your honeypots. `https://github.com/0x4D31/honeybits`. [Accedido: 20-SEP-2019].

[13] GitHub. Github - 0x4d31/honeyku: A heroku-based web honeypot that can be used to create and monitor fake http endpoints (i.e. honeytokens). `https://github.com/0x4D31/honeyku`. [Accedido: 20-SEP-2019].

[14] Heroku. Cloud application platform | heroku. https://www.heroku.com. [Accedido: 13-OCT-2019].

[15] GitHub. Github - secureworks/dcept: A tool for deploying and detecting use of active directory honeytokens. https://github.com/secureworks/dcept. [Accedido: 20-SEP-2019].

[16] NeoAttack. ¿qué es un framework y para que sirve? - neo wiki | neoattack. https://neoattack.com/neowiki/framework/. [Accedido: 27-NOV-2020].

[17] Germán Escobar. El protocolo http. https://blog.makeitreal.camp/el-protocolo-http/. [Accedido: 29-AGO-2020].

[18] Wikipedia. Diagrama de despliegue. https://es.wikipedia.org/wiki/Diagrama_de_despliegue. [Accedido: 25-SEP-2020].

[19] MDN Web Docs. Service worker api | mdn. https://developer.mozilla.org/es/docs/Web/API/Service_Worker_API. [Accedido: 19-SEP-2020].

[20] MDN Web Docs. Using service workers | mdn. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers. [Accedido: 19-SEP-2020].

[21] W3Schools. Html input types. https://www.w3schools.com/html/html_form_input_types.asp. [Accedido: 30-AGO-2020].

[22] Wikipedia. Expresión regular - wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Expresion_regular. [Accedido: 05-DIC-2020].

[23] MDN Web Docs. Métodos de petición http - http | mdn. https://developer.mozilla.org/es/docs/Web/HTTP/Methods. [Accedido: 05-DIC-2020].

[24] MISP project. Misp - open source threat intelligence platform & open standards for threat information sharing (formely known as malware information sharing platform). https://www.misp-project.org. [Accedido: 02-NOV-2020].

[25] OWASP Foundation. Owasp zap. https://www.zaproxy.org/. [Accedido: 01-NOV-2020].

# Appendices

## A. Funcionamiento de la herramienta

A continuación se mostrarán diferentes evidencias que ilustran el funcionamiento de la herramienta desarrollada en los diferentes escenarios donde se considera que un usuario está generando un abuso sobre los Honeytokens desplegados.

### A.1. Escenario 1: Modificación del valor de un input oculto

Al ingresar en la aplicación, se ingresa en el formulario "*Contacto*" que aparece disponible en la página principal del sitio y donde se tiene configurado un Honeytoken como **input oculto**.

Dentro de la pantalla de "Contacto", al utilizar la herramienta *Inspector* del navegador se puede visualizar el **input oculto** mencionado anteriormente, el cual se encuentra dentro del formulario que se presenta en la página, como se muestra a continuación:


Figura 15: Captura de input oculto dentro de una página web

Para el Honeypot, es esperable que este input oculto retorne de la forma que está, es decir, sin alterar el valor dentro del parámetro "*value*". Por lo tanto, se modifica este valor presente por el de *"ataque"* para analizar el comportamiento del Honeypot:

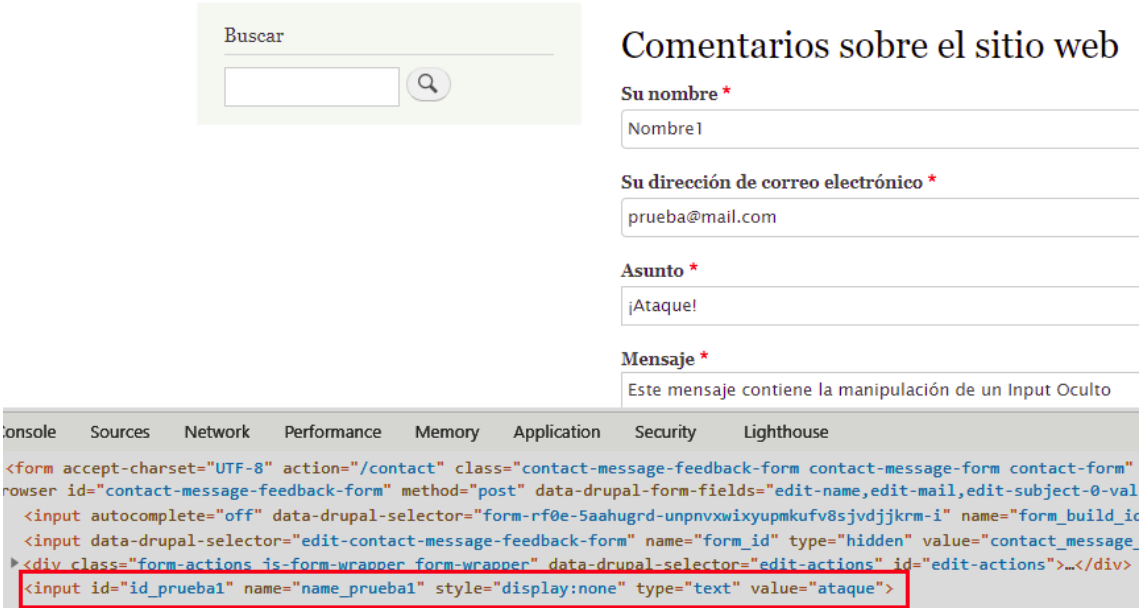


Figura 16: Captura modificación input oculto sobre la página web

Una vez que el Honeypot recibe el input oculto modificado, registra información de dicha modificación dentro de tres lugares: log del sistema, log con información de las sesiones creadas para los usuarios y log con información de los Honeytokens modificados. A continuación se muestra como es presentada la información por el sistema:

```
[2020-10-28 22:32:47,863][INFO][Honeypot]: executing detect_honeytoken_manipulation(...)
[2020-10-28 22:32:47,868][INFO][SessionCookie]: executing has_message_session_id(...)
[2020-10-28 22:32:47,872][INFO][UndefUserCookie]: executing has_message_undef_user_id(...)
[2020-10-28 22:32:47,873][INFO][HTMLInputInstancia]: executing is_manipulated(...) method
[2020-10-28 22:32:47,875][INFO][HTMLInputInstancia]: Manipulación detectada: CHANGED_VALUE
[2020-10-28 22:32:47,883][INFO][Honeypot]: Sesión: e15879ec-81b9-46e4-9206-e79327456b4e
[2020-10-28 22:32:47,884][INFO][JSONLog]: executing log_user_information(...)
```

Figura 17: Captura Log del Sistema al momento de la modificación de un input oculto


Figura 18: Captura log con información de sesiones

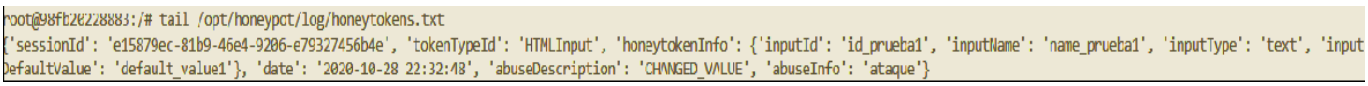


Figura 19: Captura log con información de Honeytokens al momento de la modificación de un input oculto

Como se ha mencionado anteriormente, el Honeypot asigna una sesión una vez que detecta la manipulación de un Honeytoken. En este caso se puede apreciar que se le asignó la sesión con el valor "e15879ec-81b9-46e4-9206-e79327456b4e". Dicho valor es enviado al cliente web mediante una cookie llamada "**session_cookie**":

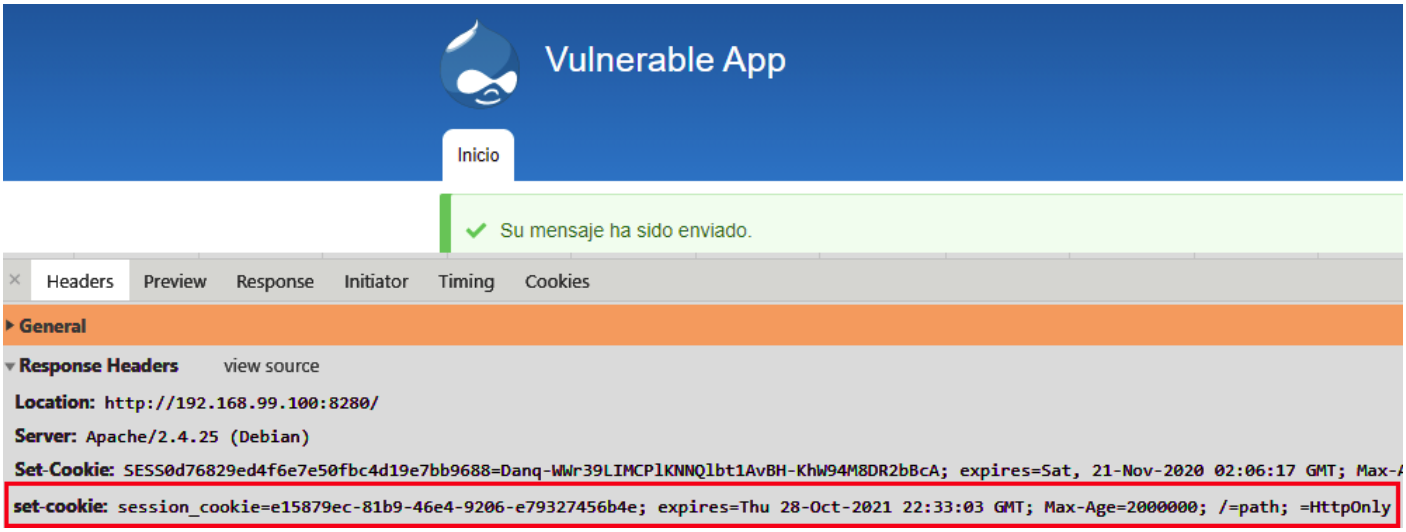


Figura 20: Captura modificación input oculto

A su vez, se puede apreciar que el cliente no tuvo alteraciones en la respuesta modificada por el Honeypot, solamente agrega la nueva cookie de sesión brindado por el Honeypot para darle seguimiento a dicho cliente.

## A.2. Escenario 2: Eliminación de input oculto

En este escenario se elimina el **input oculto** que fue modificado anteriormente utilizando la misma sesión que anteriormente modificó el mismo input. Para esto, se ingresa nuevamente en el formulario "*Contacto*" y se utiliza la herramienta *Inspector* del navegador para visualizar el **input oculto** a eliminar:


Figura 21: Captura de input oculto dentro de una página web

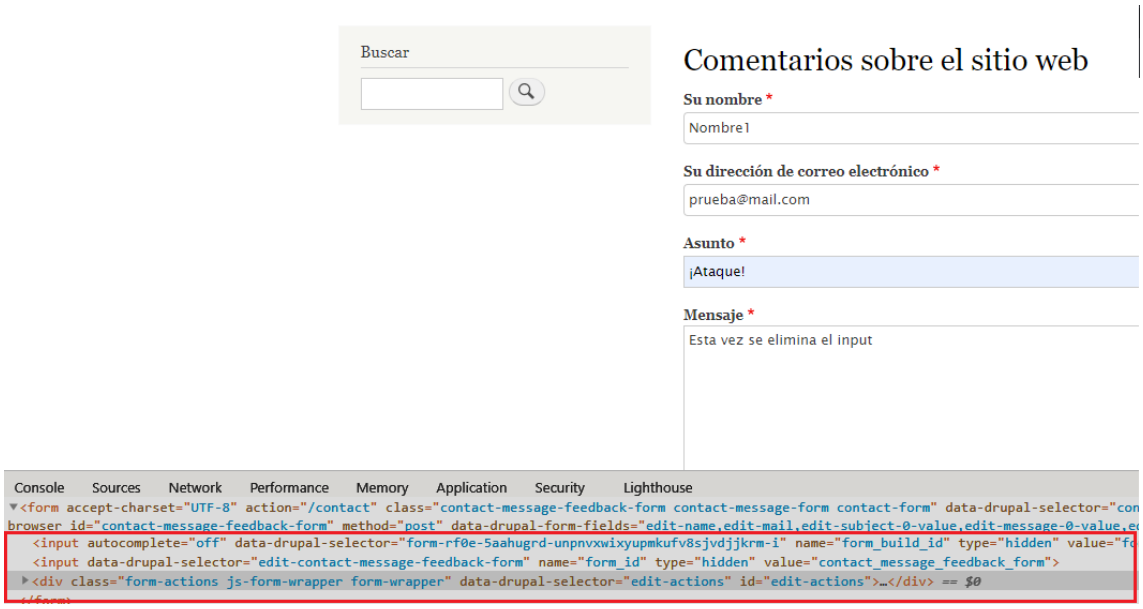


Figura 22: Captura de eliminación de input oculto

El Honeypot recibe el request sin el input oculto, por lo cual registra este evento en el log del sistema y log con información de los Honeytokens modificados, no ingresa información en el log de sesiones dado que la misma ya existe:

```
192.168.99.1:54503: clientconnect
[2020-10-28 23:16:05,283][INFO][Honeypot]: executing detect_honeytoken_manipulation(...)
[2020-10-28 23:16:05,287][INFO][SessionCookie]: executing has_message_session_id(...)
[2020-10-28 23:16:05,288][INFO][HTMLInputInstancia]: executing is_manipulated(...) method
[2020-10-28 23:16:05,290][INFO][HTMLInputInstancia]: Manipulación detectada: DELETED_OR_EMPTY_VALUE
[2020-10-28 23:16:05,292][INFO][JSONLog]: executing log_user_information(...)
```

Figura 23: Captura Log del Sistema al momento de la eliminación de un input oculto

```
root@98fb20228883:/# tail /opt/honeypot/log/attacker.txt
{'ipAddress': '::ffff:192.168.99.1', 'userAgent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.111 Safari/537.36', 'date': '2020
-10-28 22:32:48', 'sessionId': 'e15879ec-81b9-46e4-9206-e79327456b4e'}
```

Figura 24: Captura log con información de sesiones

```
root@98fb20228883:/# tail /opt/honeypot/log/honeytokens.txt
{'sessionId': 'e15879ec-81b9-46e4-9206-e79327456b4e', 'tokenTypeId': 'HTMLInput', 'honeytokenInfo': {'inputId': 'id_prueba1', 'inputName': 'name_prueba1', 'inputType': 'text', 'input
DefaultValue': 'default_value1'}, 'date': '2020-10-28 22:32:48', 'abuseDescription': 'CHANGED_VALUE', 'abuseInfo': 'ataque'},
{'sessionId': 'e15879ec-81b9-46e4-9206-e79327456b4e', 'tokenTypeId': 'HTMLInput', 'honeytokenInfo': {'inputId': 'id_prueba1', 'inputName': 'name_prueba1', 'inputType': 'text', 'input
DefaultValue': 'default_value1'}, 'date': '2020-10-28 23:16:05', 'abuseDescription': 'DELETED_OR_EMPTY_VALUE', 'abuseInfo': 'ataque'}
```

Figura 25: Captura log con información de Honeytokens al momento de la eliminación de un input oculto

Por último se puede apreciar como el cliente no percibe modificaciones en la página presentada por la aplicación y modificada por el Honeypot:


Figura 26: Captura modificación input oculto

## A.3. Escenario 3: Modificación del valor de una cookie

Como se menciono anteriormente, los Honeytokens de tipo **cookie** solamente detectan la modificación del valor de la cookie como abuso. Por lo tanto, en este escenario de muestra la modificación de la **cookie** configurada para que aparezca dentro del formulario de "*Contacto*", como se estuvo trabajando en los ejemplos pasados. También, para este caso se utilizará una nueva sesión en las pruebas.

Como se aprecia en las siguientes imágenes, al acceder a la URL */contact* se presenta la cookie **cookie_prueba_contact**, la cual no se encuentra al acceder a otras URLs.

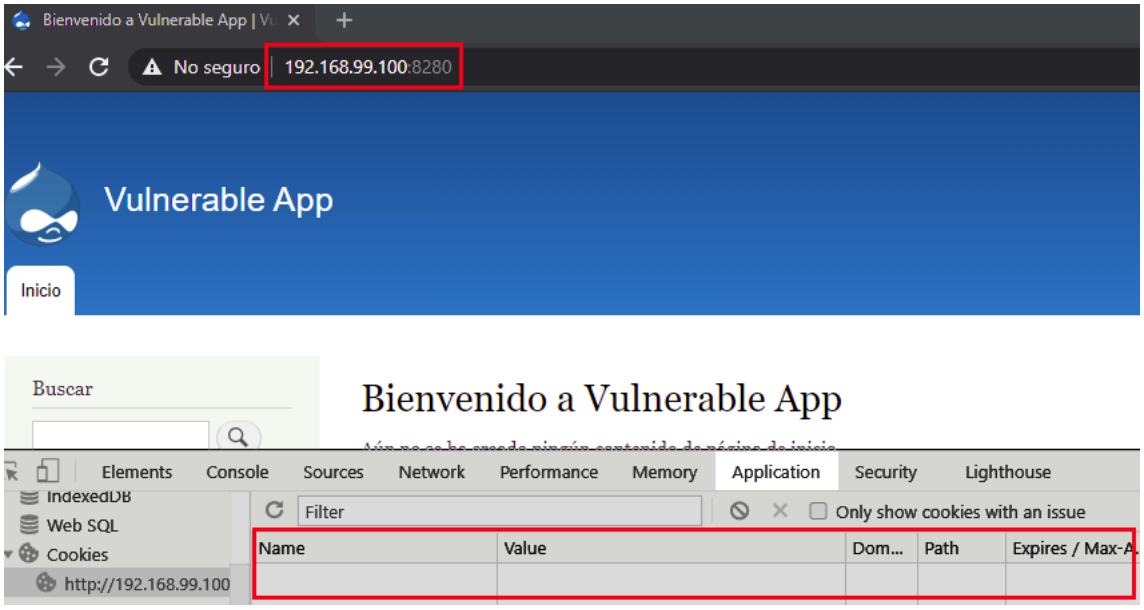


Figura 27: Captura de URL index donde no se encuentra la cookie

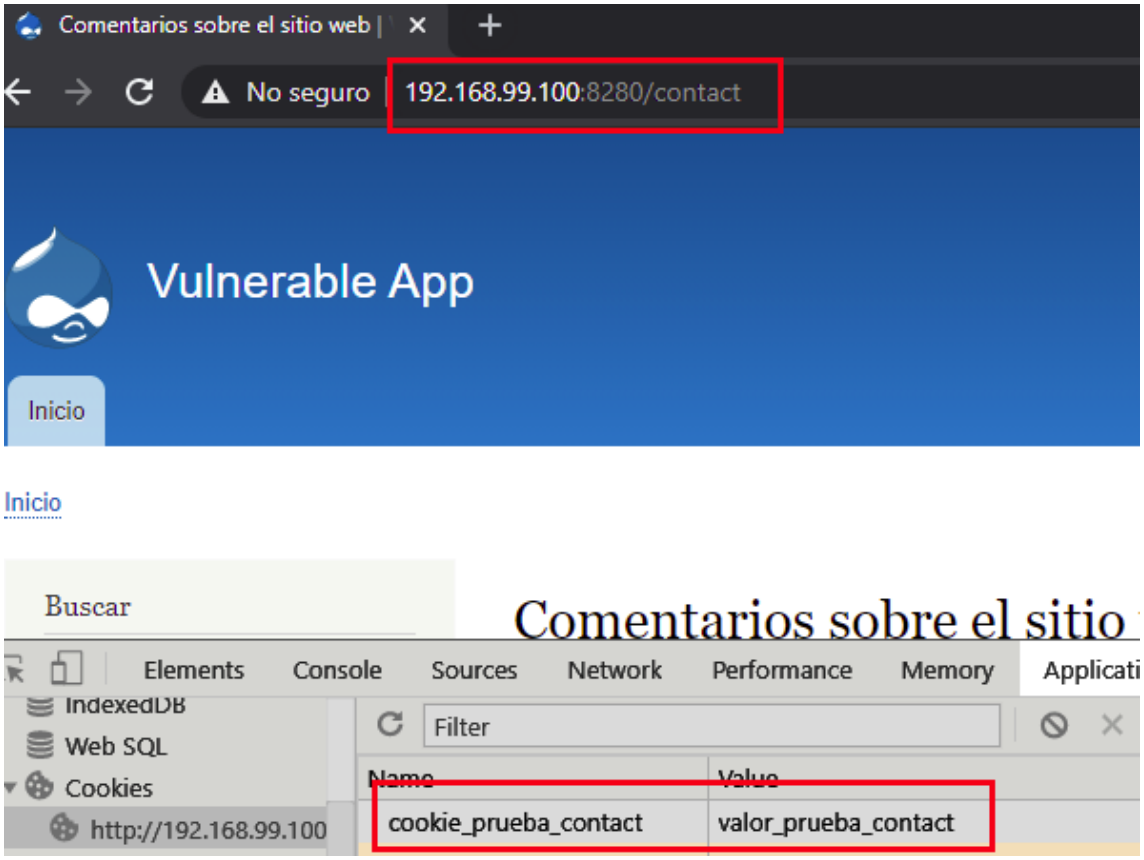


Figura 28: Captura de URL "/contact" donde se encuentra la cookie

Dado que solamente el cliente web envía esta cookie cuando se accede a la URL */contact*, mediante la herramienta **OWASP ZAP** ([25]), la cual es un proxy utilizado para interceptar el tráfico entre el cliente web y la aplicación, se intercepta la solicitud generada desde el navegador hacia la aplicación, donde se puede apreciar el envio de la cookie por parte del cliente:


Figura 29: Captura de OWASP ZAP enviando la cookie hacia la aplicación

Esta cookie, será modificada con el valor *"ataque"*:


Figura 30: Captura de OWASP ZAP donde se modifica el valor de la cookie

Como sucede en el **Escenario 2**, el Honeypot al recibir el request con la cookie modificada, registra este evento en el log del sistema y log con información de los Honeytokens modificados, ingresando también información en el log de sesiones debibo a que se está utilizando un nuevo cliente web:

```
root@98fb20228883:/# tail -n1 /opt/honeypot/log/attacker.txt
{'ipAddress': '::ffff:192.168.99.1', 'userAgent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
/537.36 (KHTML, like Gecko) Chrome/86.0.4240.193 Safari/537.36', 'date': '2020-11-17 20:59:45', 'session
Id': '627563b3-dc9b-406e-b2ef-3d2679a48c66'}
```

Figura 31: Captura log con información de sesiones

```
root@98fb20228883:/# tail -n1 /opt/honeypot/log/honeytokens.txt
{'sessionId': '627563b3-dc9b-406e-b2ef-3d2679a48c66', 'tokenTypeId': 'Cookie', 'honeytokenInfo': {'cooki
eName': 'cookie_prueba_contact', 'cookieValue': 'valor_prueba_contact'}, 'date': '2020-11-17 20:59:45',
root@98fb20228883:/#
```

Figura 32: Captura log con información de Honeytokens al momento de la modificación de la cookie

Por último, se puede apreciar como al cliente llegue una nueva **cookie de sesión** con el nuevo valor generado:

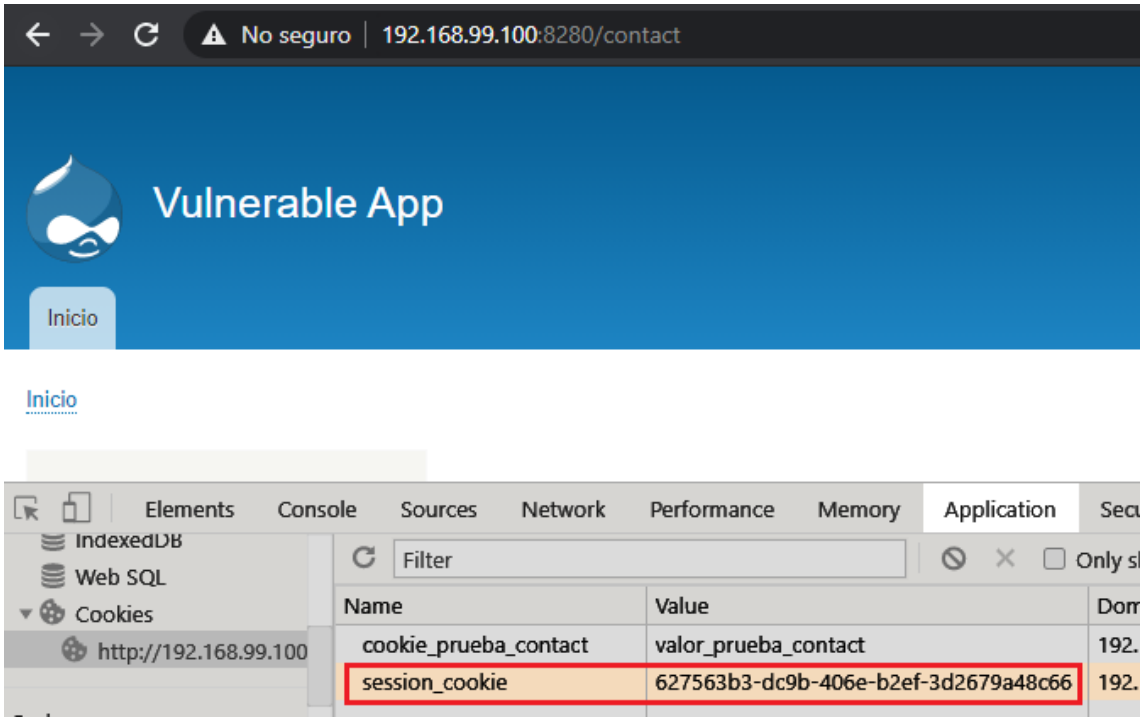


Figura 33: Nueva cookie de sesión almacenada por el cliente web