Burp Suite Plugins/Extender

https://www.linkedin.com/in/joas-antonio-dos-santos

Sumary

Setup Enviroment Development

https://www.jetbrains.com/help/pycharm/installation-guide.html

https://www3.ntu.edu.sg/home/ehchua/programming/howto/netbeans_howto.html

https://www.youtube.com/watch?v=vt7_6HwCFOU

https://kotlinlang.org/

https://www.eclipse.org/downloads/packages/installer

https://www.youtube.com/watch?v=N-wXTRpR03U

First of all you'll need an IDE. Some popular options are: IntelliJ IDEA, Netbeans, and Eclipse.

Java Config

Create a new project, and create a package called "burp". Next you'll need to copy in the interface files which you can export from Burp at Extender / APIs / Save interface files. Save the interface files into the folder that was created for the burp package.

Now that you have the general environment set up you'll need to create the actual extension file. Create a new file called BurpExtender.java (or a new class called BurpExtender, if your IDE makes the files for you) and paste in the following code:

```
package burp;
public class BurpExtender implements IBurpExtender
{
    public void registerExtenderCallbacks (IBurpExtenderCallbacks callbacks)
    {
        // your extension code here
    }
}
```

This example does nothing at all, but will compile and can be loaded into Burp after you generate a JAR file from your IDE - it will usually be in a build or dist directory. In Burp, go to the Extender tool, and the Extensions tab, and add a new extension. Select the extension type "Java", and specify the location of your JAR file. This should be loaded into Burp with no errors.

https://portswigger.net/burp/extender/api/burp/iburpextender.html

https://github.com/iSECPartners/hiccupy/blob/master/src/burp/IBurpExtender.java

https://portswigger.net/burp/documentation/desktop/tools/extender

https://www.youtube.com/watch?v=wR1ENja0lI0

Python Config

Burp relies on Jython to provide its Python support. You will need to download the "Standalone Jar" version and configure Burp with its location (at Extender / Options / Python environment).

Now create a new file with any name you like, ending in '.py', and add the following content to that file:

```
from burp import IBurpExtender
class BurpExtender(IBurpExtender):
    def registerExtenderCallbacks( self, callbacks):
        # your extension code here
        return
```

Then go to the Extensions tab, and add a new extension. Select the extension type "Python", and specify the location of your file.

https://forum.portswigger.net/thread/burp-extension-python-import-error-c0f9ab13

https://wiki.owasp.org/images/9/9f/Extending-Burp-with-Python.pptx

https://portswigger.net/burp/documentation/desktop/tools/extender

https://www.youtube.com/watch?v=4f05lNULX1I

IBurpExtender

All extensions must implement this interface. Implementations must be called BurpExtender, in the package burp, must be declared public, and must provide a default (public, no-argument) constructor.

All extensions must implement this interface.

▪ BurpExtender must implement this interface with one method

– void registerExtenderCallbacks(IBurpExtenderCallbacks callbacks)

▪ Burp invokes registerExtenderCallbacks()method when na extension is loaded

https://portswigger.net/burp/extender/api/burp/iburpextendercallbacks.html

IBurpExtenderCallbacks

public interface IBurpExtenderCallbacks

This interface is used by Burp Suite to pass to extensions a set of callback methods that can be used by extensions to perform various actions within Burp. When an extension is loaded, Burp invokes its registerExtenderCallbacks() method and passes an instance of the IBurpExtenderCallbacks interface. The extension may then invoke the methods of this interface as required in order to extend Burp's functionality.

https://portswigger.net/burp/extender/api/burp/iburpextendercallbacks.html

IExtensionHelpers

public interface IExtensionHelpers

This interface contains a number of helper methods, which extensions can use to assist with various common tasks that arise for Burp extensions. Extensions can call IBurpExtenderCallbacks.getHelpers to obtain an instance of this interface.

https://portswigger.net/burp/extender/api/burp/iextensionhelpers.html

All Packages Burp Suite

- Interface
- Description

IBurpCollaboratorClientContext
This interface represents an instance of a Burp Collaborator client context, which can be used to generate Burp Collaborator payloads and poll the Collaborator server for any network interactions that result from using those payloads.

IBurpCollaboratorInteraction
This interface represents a network interaction that occurred with the Burp Collaborator server.

IContextMenuFactory
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerContextMenuFactory() to register a factory for custom context menu items.

IContextMenuInvocation
This interface is used when Burp calls into an extension-provided IContextMenuFactory with details of a context menu invocation.

ICookie
This interface is used to hold details about an HTTP cookie.

IExtensionStateListener
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerExtensionStateListener() to register an extension state listener.

IHttpHeader
This interface is used to hold details about an HTTP/2 header.

IHttpListener
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerHttpListener() to register an HTTP listener.

IHttpRequestResponse
This interface is used to retrieve and update details about HTTP messages.

IHttpRequestResponsePersisted
This interface is used for an IHttpRequestResponse object whose request and response messages have been saved to temporary files using IBurpExtenderCallbacks.saveBuffersToTempFiles().

IHttpRequestResponseWithMarkers
This interface is used for an IHttpRequestResponse object that has had markers applied.

IHttpService
This interface is used to provide details about an HTTP service, to which HTTP requests can be sent.

IInterceptedProxyMessage
This interface is used to represent an HTTP message that has been intercepted by Burp Proxy.

IIntruderAttack
This interface is used to hold details about an Intruder attack.

IIntruderPayloadGenerator
This interface is used for custom Intruder payload generators.

IIntruderPayloadGeneratorFactory
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerIntruderPayloadGeneratorFactory() to register a factory for custom Intruder payloads.

IIntruderPayloadProcessor
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerIntruderPayloadProcessor() to register a custom Intruder payload processor.

IMenuItemHandler Deprecated
Use IContextMenuFactory instead.

IMessageEditor
This interface is used to provide extensions with an instance of Burp's HTTP message editor, for the extension to use in its own UI.

IMessageEditorController
This interface is used by an IMessageEditor to obtain details about the currently displayed message.

IMessageEditorTab
Extensions that register an IMessageEditorTabFactory must return instances of this interface, which Burp will use to create custom tabs within its HTTP message editors.

IMessageEditorTabFactory
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerMessageEditorTabFactory() to register a factory for custom message editor tabs.

IParameter
This interface is used to hold details about an HTTP request parameter.

IProxyListener
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerProxyListener() to register a Proxy listener.

IRequestInfo
This interface is used to retrieve key details about an HTTP request.

IResponseInfo
This interface is used to retrieve key details about an HTTP response.

IResponseKeywords
This interface is used to represent the counts of keywords appearing in a number of HTTP responses.

IResponseVariations
This interface is used to represent variations between a number HTTP responses, according to various attributes.

IScanIssue
This interface is used to retrieve details of Scanner issues.

IScannerCheck
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerScannerCheck() to register a custom Scanner check.

IScannerInsertionPoint
This interface is used to define an insertion point for use by active Scanner checks.

IScannerInsertionPointProvider
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerScannerInsertionPointProvider() to register a factory for custom Scanner insertion points.

IScannerListener
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerScannerListener() to register a Scanner listener.

IScanQueueItem
This interface is used to retrieve details of items in the Burp Scanner active scan queue.

IScopeChangeListener
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerScopeChangeListener() to register a scope change listener.

ISessionHandlingAction
Extensions can implement this interface and then call IBurpExtenderCallbacks.registerSessionHandlingAction() to register a custom session handling action.

ITab
This interface is used to provide Burp with details of a custom tab that will be added to Burp's UI, using a method such as IBurpExtenderCallbacks.addSuiteTab().

ITempFile
This interface is used to hold details of a temporary file that has been created via a call to IBurpExtenderCallbacks.saveToTempFile().

ITextEditor
This interface is used to provide extensions with an instance of Burp's raw text editor, for the extension to use in its own UI.

Snoopysecurity Burp Suite Extensions List

https://github.com/snoopysecurity/awesome-burp-extensions

Bug Bounty Plugins

https://burpbounty.com/ - Burp Bounty Pro is a Burp Suite Pro extension that improves the active and passive scanner by utilizing advanced and customized vulnerability profiles through a very intuitive graphical interface.

Active scanner ++ - Active scanner ++ is commonly Used Burp Extension Which Help Us to Different Security Issue Against Our Targeted Website Such As SSTI, Host header Injection, cache poisoning, DNS rebinding, XML input handling ETC.

Param-miner - Param-miner is Awesome extension Which Help A Tester To Validate issues hidden parameter, unlinked parameters. It's particularly useful for finding web cache poisoning vulnerabilities.

Collaborator Everywhere - Collaborator Everywhere This I recommend To Use if Your Looking For Blind SSRF Types Of Security Issue By adding Unique Headers On request designed to reveal backend systems by causing pingbacks to Burp Collaborator. This We can Defile as Out of band Response From Server . This Extension Also help to find Blind SQLI , Command Injection etc.

Logger ++ - Logger++ is a multithreaded logging extension for Burp Suite which help a tester to filter highlight interesting entries or filter logs to only that match the filter . A built in grep tool allows the logs to be searched to locate entries which match a specified pattern, and extract the values of the capture groups.

Autorize - Autorize Is Extension Based On finding Authorization Issue Against Our Targeted Website This is one of the time-consuming tasks we can make it automate in a web application penetration test phase such as it help us to Find out Authorization Related Issue , IDOR Insecure Direct access control . This Help Us to validate issue based On extension the cookies of a low privileged user and navigate the website with a high privileged user.

Retire.js - Retire.js It help us to extract all JS Files from Our Targeted Website This help us to validate all js library based On vulnerable Version and We can easily Verify Which JS Library They Are using and we can easily validate security issue According to Library Version .

Json Beautifier - This extension help Us to read Json and js file In good manner It is diffcult for web application security tester to analyse the JS files which are compressed to increase the loading speed. This extension help us to verify Resources based On JSON or JS

XSS Validator - XSS Validator Is Burp Extension Which help us to validate XSS cross Site Scripting Related Security loopholes Against Our targeted Website using different regex value This extension send Response to server Which based on Phantom.js and/or Slimer.js

Backslash-powered-scanner - This extension Is based On Burp Active Scanner which commonly validate Known and unknown classes of Server Side Injection Security issue.

403Bypasser - After a lot of effort, you come across a potentially vulnerable end-point but receive a 403-status code. Believe me there's nothing more disheartening than receiving a 403 Response Code.

BurpSuiteAutoCompletion - Have trouble searching for payloads? What if an addon could do it for you? Wouldn't it be better if an extension could autocomplete your payloads? BurpSuiteAutoCompletion is one such addon that can help you autocomplete payloads. It works only in the Repeater and the Intruder tabs. For more information from the authors and how to use it.

Cloud Storage Tester - More and more companies are opting for cloud-based services. The major leaders in this domain are AWS, Google and Microsoft Azure. What if we are a newbie and just started with web application hacking? It might be a little tough to test the services in the cloud when you hardly know what they are and how to test them.

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

You can install BApps directly within Burp, via the BApp Store feature in the Burp Extender tool. You can also download them from here, for offline installation into Burp.

Community and Professional Plugins

.NET Beautifier

Masks verbose parameter details in .NET requests.

5GC API Parser

Allows you to assess 5G core network functions by parsing OpenAPI 3.0, and generate requests for intrusion testing purposes.

Add Custom Header

Add or update custom HTTP headers from session handling rules. Useful for JWT.

Additional CSRF Checks

Performs additional checks for CSRF vulnerabilities in a semi-automated manner.

Adhoc Payload Processors

Generate payload processors on the fly - without having to create individual extensions.

AES Killer, decrypt AES traffic on the fly

Decrypt AES traffic on the fly

Anti-CSRF Token From Referer

Automatically takes care of anti-CSRF tokens by fetching them from the referer and replacing them in requests.

Auth Analyzer

This Burp Extension helps you to find authorization bugs by repeating Proxy requests with self defined headers and tokens.

Attack Surface Detector

Use static analysis to identify web app endpoints by parsing routes and identying parameters.

Authentication Token Obtain and Replace

Helps automated scanning accessing/refreshing tokens, replacing tokens in XML and JSON body,replacing tokens in cookies.

Auto-Drop Requests

This extension allows you to automatically Drop requests that match a certain regex.

AutoRepeater

Automatically repeat requests, with replacement rules and response diffing.

AWS Signer

Signs requests with AWS Signature Version 4

Blazer

Generates and fuzzes custom AMF messages.

Brida, Burp to Frida bridge

A bridge between Burp Suite and Frida to help test Android applications.

BugPoC

Send raw HTTP requests to BugPoC.com

Burp Bounty - Scan Check Builder

This BurpSuite extension allows you, in a quick and simple way, to improve the active and passive burpsuite scanner by means of personalized rules through a very intuitive graphical interface.

Burp2Telegram

Push notifications to Telegram bot on BurpSuite response

BurpSmartBuster

Looks for files, directories and file extensions based on current requests received by Burp Suite.

Bypass WAF

Adds headers useful for bypassing some WAF devices.

Command Injection Attacker

Customizable payload generator to detect and exploit command injection flaws during blind testing.

Copy as PowerShell Requests

Copies the selected request(s) as PowerShell invocation(s).

CSP Auditor

Displays CSP headers for responses, and passively reports CSP weaknesses.

CSRF Token Tracker

Provides a sync function for CSRF token parameters.

Customizer

Use different themes with Burp Suite.

ExifTool Scanner

Reads metadata from various file types (JPEG, PNG, PDF, DOC, and much more) using ExifTool.

GadgetProbe

Augments Intruder to probe endpoints consuming Java serialized objects to identify classes, libraries, and library versions on remote Java classpaths.

Google Hack

Lets you run Google Hacking queries and add results to Burp's site map.

HackBar, Payload Bucket

A burp suite extension to easily insert payloads into requests.

HTTP Digest Auth

A Burp Suite extension to handle HTTP Digest Authentication, which is no more supported by Burp Suite since version 2020.7.

HTTP Request Smuggler

Helps you launch HTTP Request Smuggling attacks, supports scanning for Request Smuggling vulnerabilities and also aids exploitation by handling cumbersome offset-tweaking for you

Hunt Scanner

Passively scan for potentially vulnerable parameters.

InQL - Introspection GraphQL Scanner

InQL - A Burp Extension for GraphQL Security Testing

Intruder File Payload Generator

Allows use of file contents and filenames as Intruder payloads.

Java Deserialization Scanner

Performs active and passive scans to detect Java deserialization vulnerabilities.

Java Serial Killer

Performs Java deserialization attacks using the ysoserial payload generator tool.

Java Serialized Payloads

Generates Java serialized payloads to execute OS commands.

JCryption Handler

Analyze web applications that use JCryption

JSON Web Token Attacker

JOSEPH - JavaScript Object Signing and Encryption Pentesting Helper

JWT Editor

Edit, sign, verify, encrypt and decrypt JSON Web Tokens (JWTs).

LightBulb WAF Auditing Framework

An open source python framework for auditing WAFs and Filters.

NTLM Challenge Decoder

Decode NTLM SSP headers and extract domain/host information

Nuclei Burp Integration

Allows you to run Nuclei Scanner directly from Burp and transforms JSON results into the issues.

OAuth2 Token Grabber

Grab OAuth2 access tokens and add them to requests as a custom header.

PDF Viewer

Allows viewing of PDF files directly within Burp.

PHP Object Injection Slinger

Designed to help you find PHP Object Injection vulnerabilities on popular PHP Frameworks.

Potential Vulnerability Indicator

Checks application requests and responses for indicators of vulnerability or targets for attack

Proxy Auto Config

Automatically configures Burp upstream proxies to match desktop proxy settings.

Replicator

Helps developers replicate findings discovered in pen tests.

Same Origin Method Execution

Detects same origin method execution vulnerabilities.

San Scanner

Enumerating associated domains & services via the Subject Alt Names section of SSL certificates.

SQLi Query Tampering

Extends and adds custom Payload Generators/Processors in Burp Suite's Intruder.

SQLiPy Sqlmap Integration

Initiates SQLMap scans directly from within Burp.

SSL Scanner

Scan for SSL vulnerabilities using techniques from testssl.sh and a2sv.

Token Incrementor

Increment a token in each request. Useful for parameters like username that must be unique.

[Turbo Intruder]

Send large numbers of HTTP requests and analyze the results

[Wayback Machine]

Generate a sitemap using Wayback Machine.

[WordPress Scanner]

Find known vulnerabilities in WordPress plugins and themes using WPScan database.

[Yara]

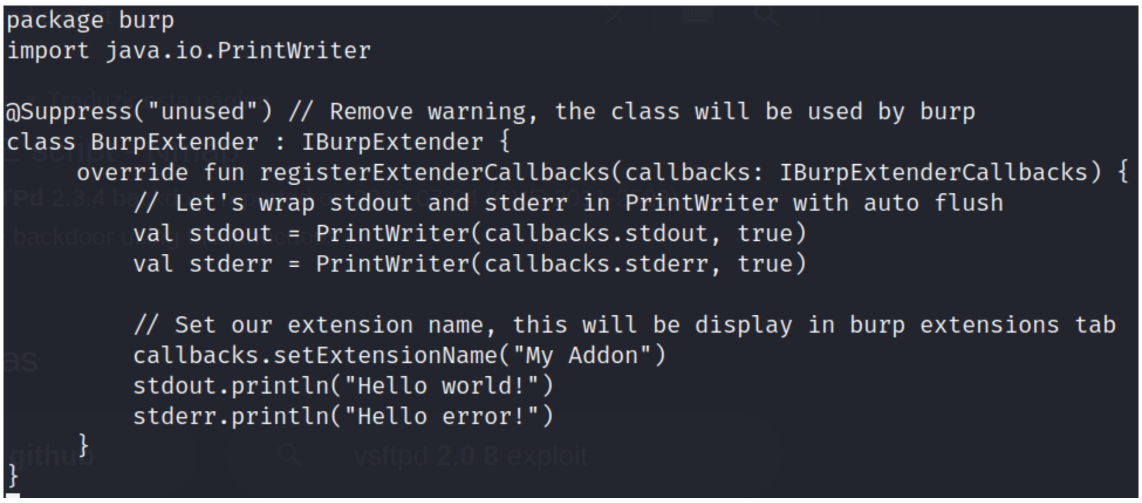
Integrates Yara scanner into Burp Suite.

[YesWeBurp]

YesWeBurp is an extension for BurpSuite allowing you to access all your https

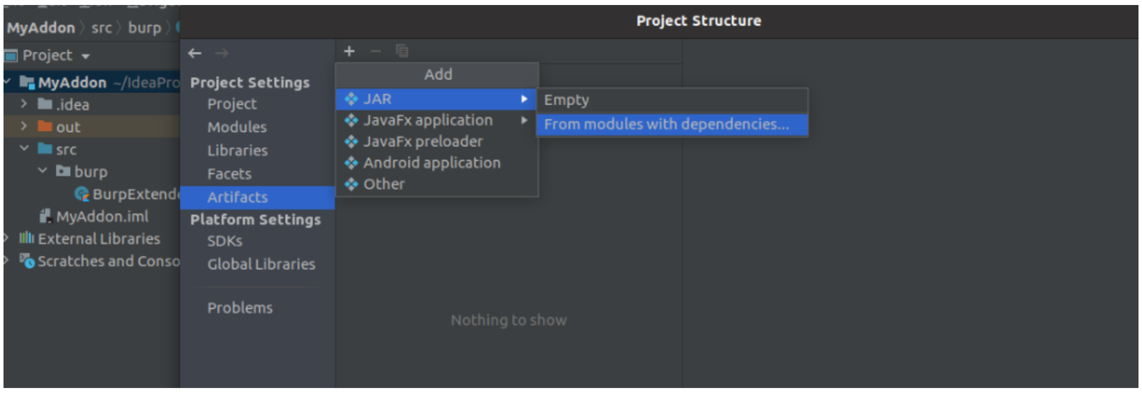
Burp Suite Plugin Development

Hello World Kotlin and Java

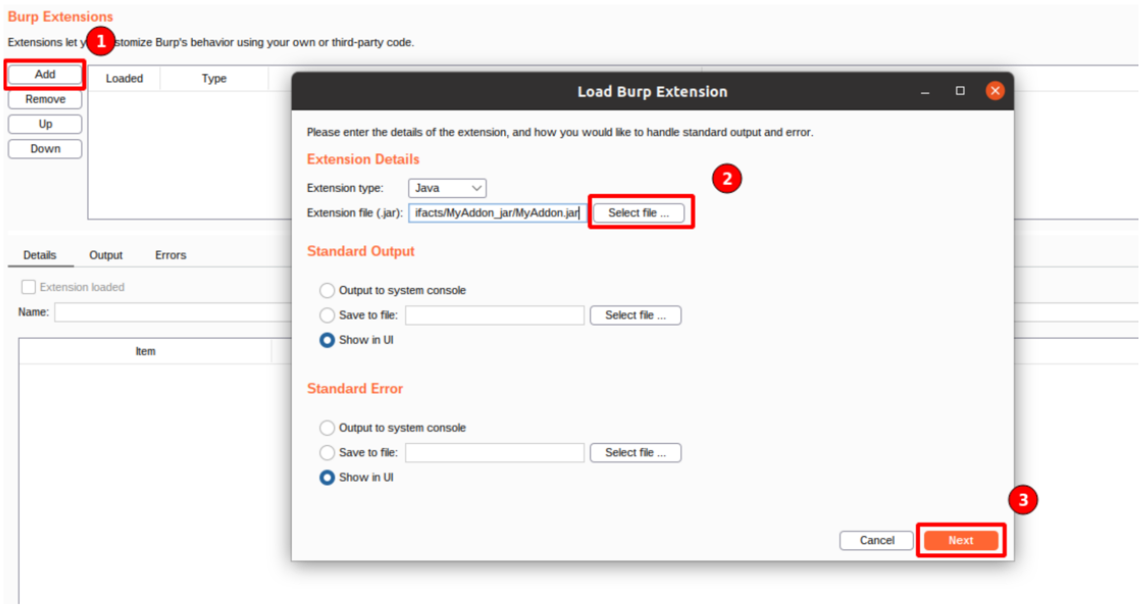


```
package burp
import java.io.PrintWriter

@Suppress("unused") // Remove warning, the class will be used by burp
class BurpExtender : IBurpExtender {
     override fun registerExtenderCallbacks(callbacks: IBurpExtenderCallbacks) {
         // Let's wrap stdout and stderr in PrintWriter with auto flush
         val stdout = PrintWriter(callbacks.stdout, true)
         val stderr = PrintWriter(callbacks.stderr, true)

         // Set our extension name, this will be display in burp extensions tab
         callbacks.setExtensionName("My Addon")
         stdout.println("Hello world!")
         stderr.println("Hello error!")
     }
}
```

Create class with Kotlin and paste code



Building Project em Jar File



Extensions let you customize Burp's behavior using your own or third-party code.

☐ Extension loaded

Name:

Please enter the details of the extension, and how you would like to handle standard output and error.

Extension Details

Extension type: Java

Extension file (.jar): ifacts/MyAddon_jar/MyAddon.jar

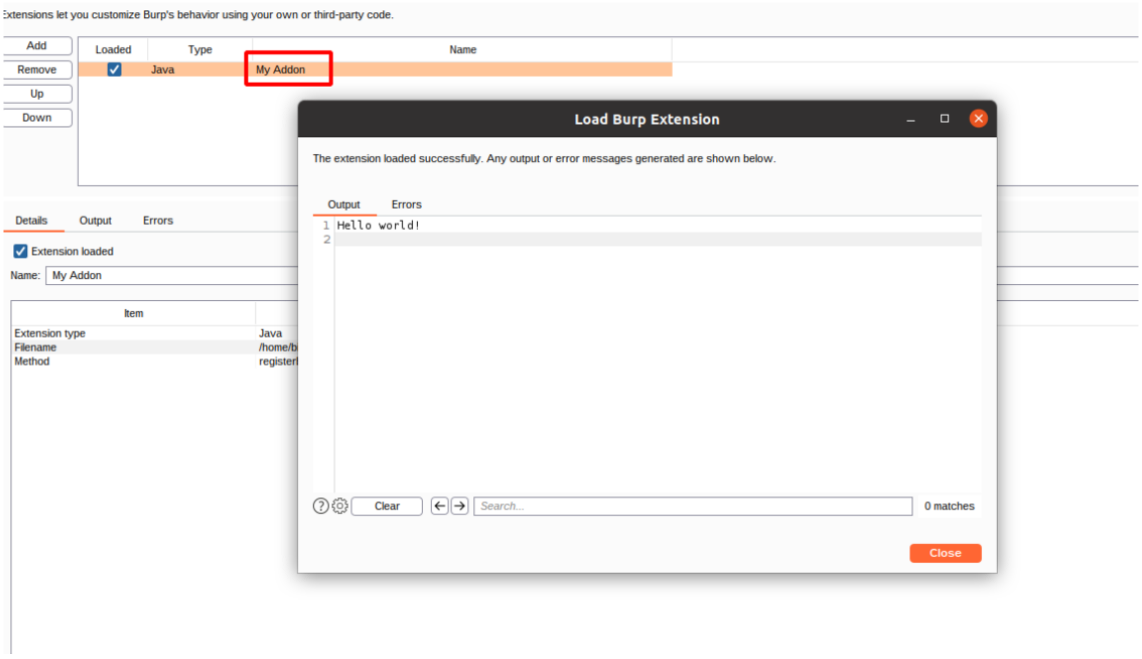
Standard Output

- ○ Output to system console
- ○ Save to file:
- [✓ Show in UI]

Standard Error

- ○ Output to system console
- ○ Save to file:
- [✓ Show in UI]

Add extension in Burp Suite



Extensions let you customize Burp's behavior using your own or third-party code.

Loaded	Type	Name
✓	Java	My Addon

☑ Extension loaded

Name: My Addon

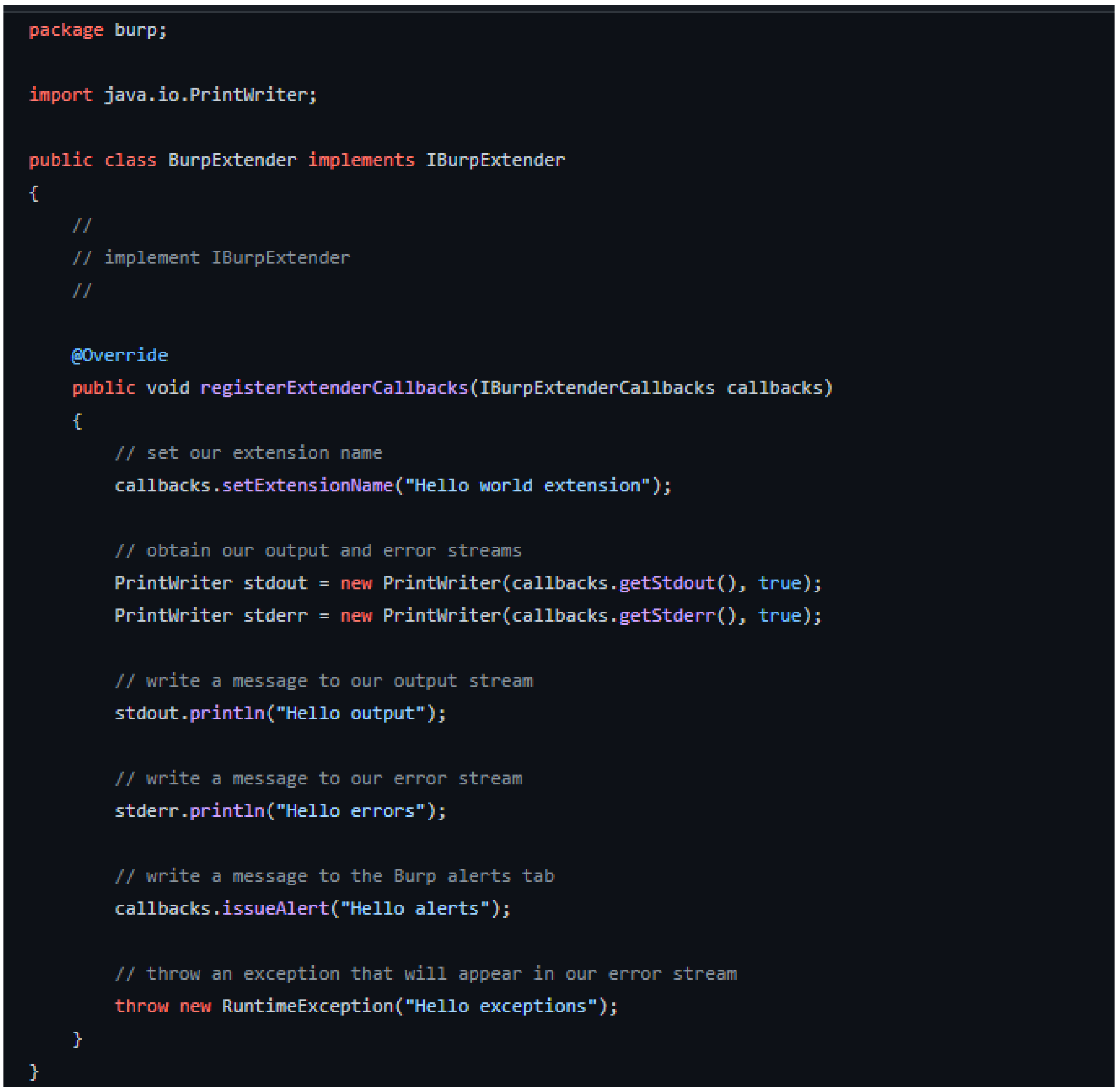
Item	
Extension type	Java
Filename	/home/b
Method	register

The extension loaded successfully. Any output or error messages generated are shown below.

```
Hello world!
```

Loading Burp Suite Extension

https://blog.yeswehack.com/yeswerhackers/tutorial/how-to-learn-write-burp-suite-extension-kotlin-setting-up/



```java
package burp;

import java.io.PrintWriter;

public class BurpExtender implements IBurpExtender
{
    //
    // implement IBurpExtender
    //

    @Override
    public void registerExtenderCallbacks(IBurpExtenderCallbacks callbacks)
    {
        // set our extension name
        callbacks.setExtensionName("Hello world extension");

        // obtain our output and error streams
        PrintWriter stdout = new PrintWriter(callbacks.getStdout(), true);
        PrintWriter stderr = new PrintWriter(callbacks.getStderr(), true);

        // write a message to our output stream
        stdout.println("Hello output");

        // write a message to our error stream
        stderr.println("Hello errors");

        // write a message to the Burp alerts tab
        callbacks.issueAlert("Hello alerts");

        // throw an exception that will appear in our error stream
        throw new RuntimeException("Hello exceptions");
    }
}
```

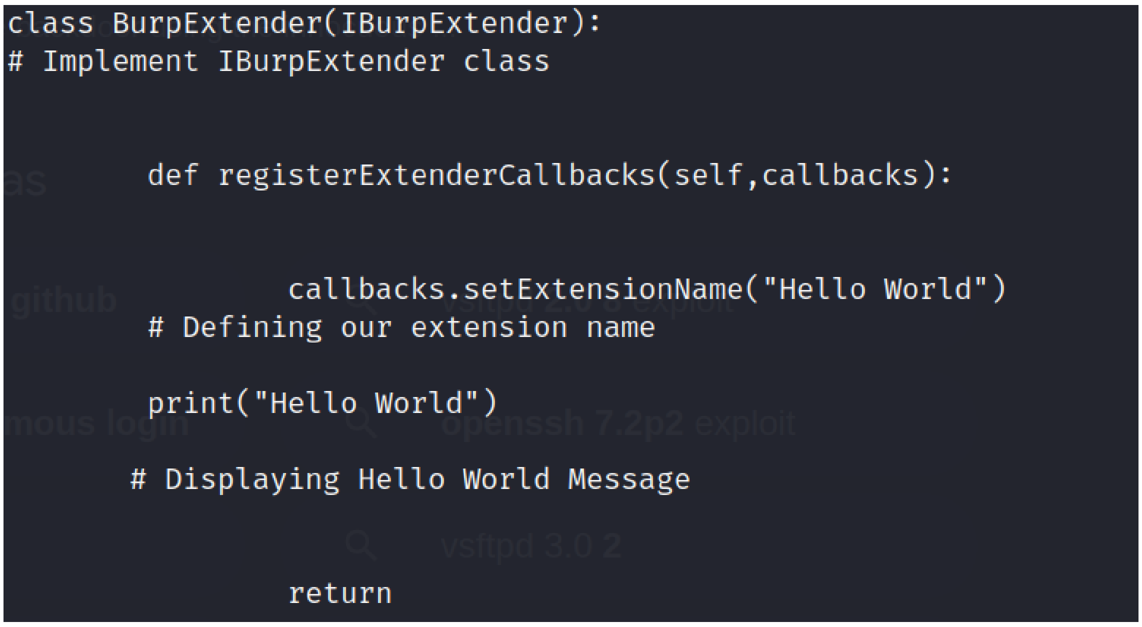
Modify and Re-Jar Extension

1. Locate the jar file: *Extender > Extensions > Select extension > Details*. bapps\ directory is located at C:\Users\yourusername\AppData\Roaming\BurpSuite\bapps
2. Backup the original jar file to a different folder, outside of bapps.
3. Change extension from .jar to .zip, extract contents, delete .zip file
4. Make your modifications
5. Re-jar: jar cvf yourJarName.jar -C extractedContentsDirectory/ .
6. Reload extension in Burp: *Extender > Extensions*, uncheck Load and check it again

Compile Extension from Source

1. Clone or download extension source code
2. Make your modifications, and create build location
3. Compile source code: javac -cp "C:\Program Files\BurpSuitePro\burpsuite_pro.jar" -d buildLocation sourceCodeLocation*.java
4. Create Jar: jar cvf yourJarName.jar buildLocation/*.class anyOtherDependencies1 anyOtherDependencies2
5. Load Jar into Burp: *Extender > Extensions*, Add, Extension type Java and locate built jar, Next, Close
6. Disable original version of extension from BApp store

Hello World Python



```
class BurpExtender(IBurpExtender):
# Implement IBurpExtender class


        def registerExtenderCallbacks(self,callbacks):


                callbacks.setExtensionName("Hello World")
        # Defining our extension name

        print("Hello World")

       # Displaying Hello World Message


                return
```

Necessary Jython in Burp Suite

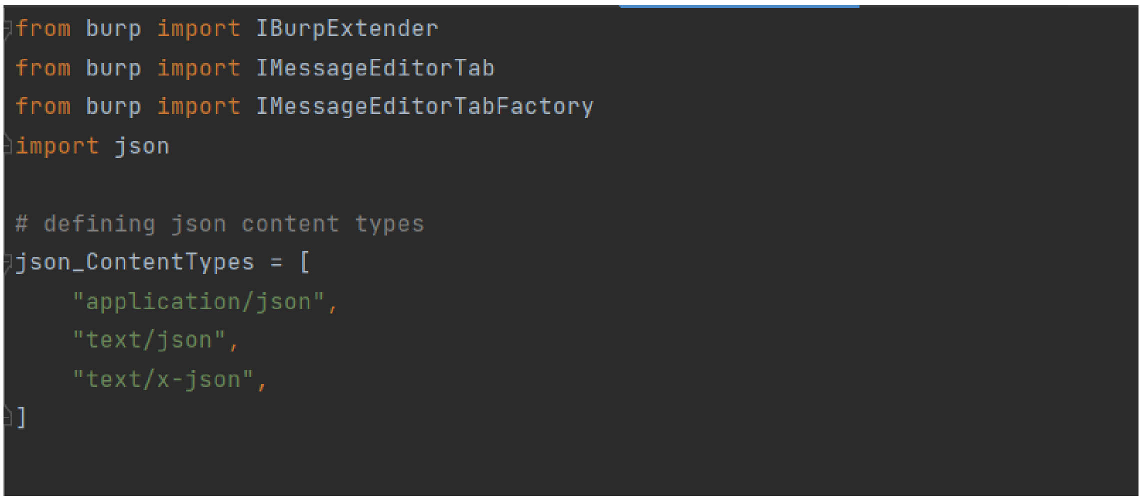
https://burpsuite.guide/runtimes/python/

The IBurpExtender class is necessary to build burp extensions. The burp will in turn look for the IBurpExtender class and then call the registerExtenderCallbacks() on this object passing in a “callback” object. The callbacks.setExtensionName is set as our extension name and the print function will print the hello world message to the screen.

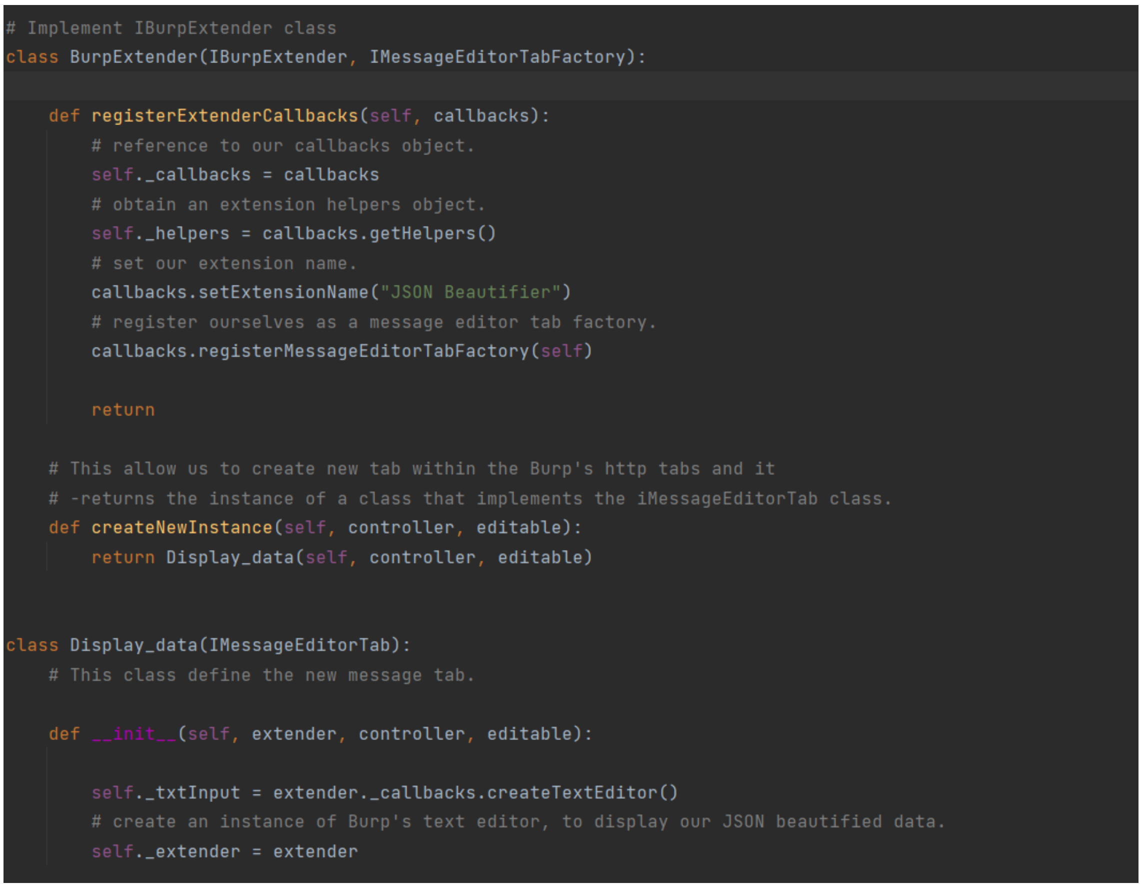
Now save the file and open it in Burp.

Add new extension: Extender > Extensions > Add > Extension Details > Extension Type: Python > Select extension file.

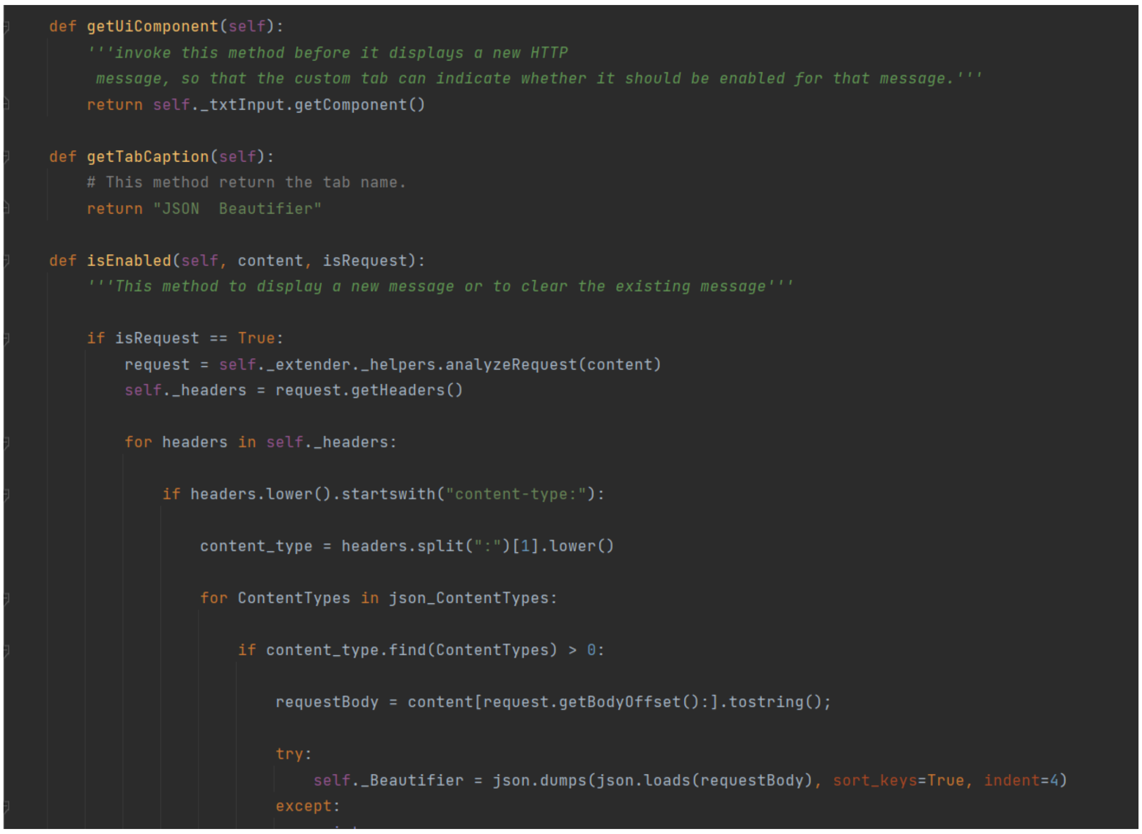
Json Beautifier



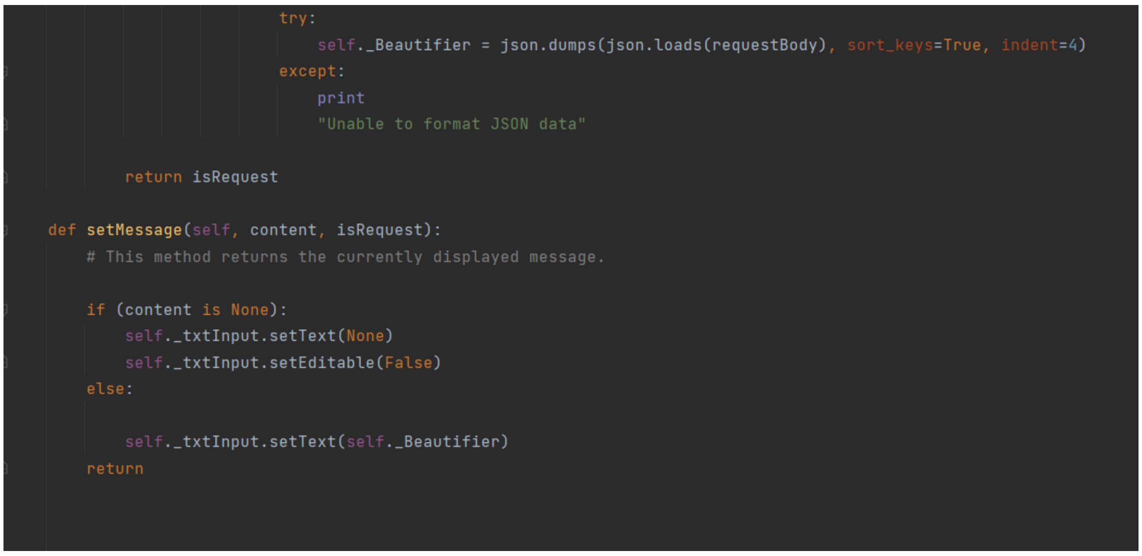
```python
from burp import IBurpExtender
from burp import IMessageEditorTab
from burp import IMessageEditorTabFactory
import json

# defining json content types
json_ContentTypes = [
    "application/json",
    "text/json",
    "text/x-json",
]
```



```python
# Implement IBurpExtender class
class BurpExtender(IBurpExtender, IMessageEditorTabFactory):

    def registerExtenderCallbacks(self, callbacks):
        # reference to our callbacks object.
        self._callbacks = callbacks
        # obtain an extension helpers object.
        self._helpers = callbacks.getHelpers()
        # set our extension name.
        callbacks.setExtensionName("JSON Beautifier")
        # register ourselves as a message editor tab factory.
        callbacks.registerMessageEditorTabFactory(self)

        return

    # This allow us to create new tab within the Burp's http tabs and it
    # -returns the instance of a class that implements the iMessageEditorTab class.
    def createNewInstance(self, controller, editable):
        return Display_data(self, controller, editable)


class Display_data(IMessageEditorTab):
    # This class define the new message tab.

    def __init__(self, extender, controller, editable):

        self._txtInput = extender._callbacks.createTextEditor()
        # create an instance of Burp's text editor, to display our JSON beautified data.
        self._extender = extender
```



```python
    def getUiComponent(self):
        '''invoke this method before it displays a new HTTP
         message, so that the custom tab can indicate whether it should be enabled for that message.'''
        return self._txtInput.getComponent()

    def getTabCaption(self):
        # This method return the tab name.
        return "JSON  Beautifier"

    def isEnabled(self, content, isRequest):
        '''This method to display a new message or to clear the existing message'''

        if isRequest == True:
            request = self._extender._helpers.analyzeRequest(content)
            self._headers = request.getHeaders()

            for headers in self._headers:

                if headers.lower().startswith("content-type:"):

                    content_type = headers.split(":")[1].lower()

                    for ContentTypes in json_ContentTypes:

                        if content_type.find(ContentTypes) > 0:

                            requestBody = content[request.getBodyOffset():].tostring();

                            try:
                                self._Beautifier = json.dumps(json.loads(requestBody), sort_keys=True, indent=4)
                            except:
```



```python
                            try:
                                self._Beautifier = json.dumps(json.loads(requestBody), sort_keys=True, indent=4)
                            except:
```

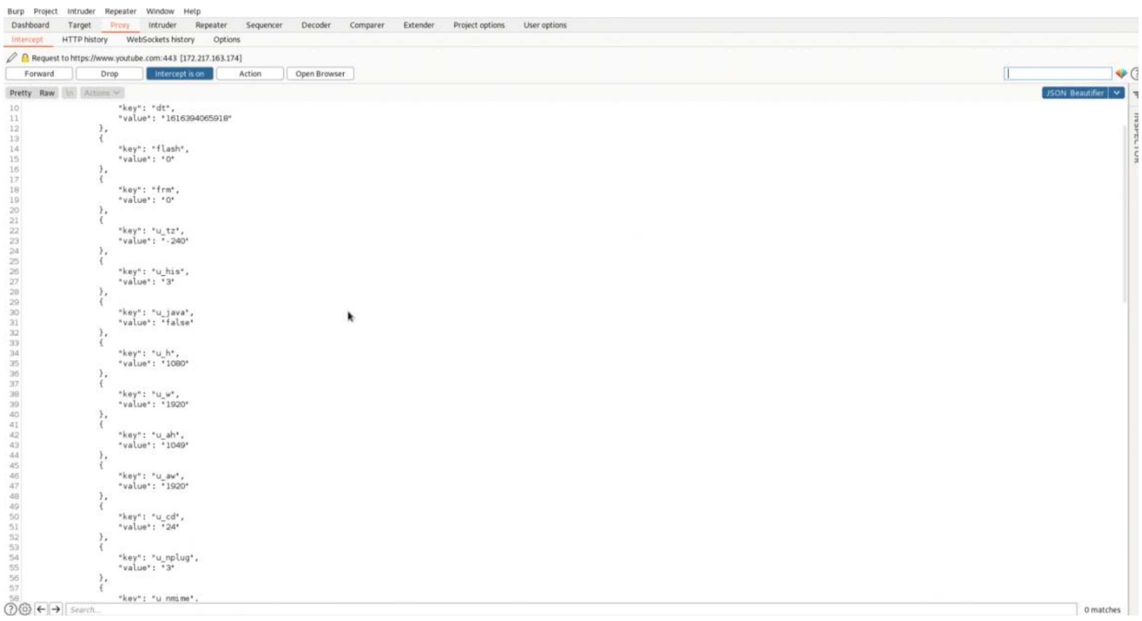
```python
                                print
                                "Unable to format JSON data"

            return isRequest

    def setMessage(self, content, isRequest):
        # This method returns the currently displayed message.

        if (content is None):
            self._txtInput.setText(None)
            self._txtInput.setEditable(False)
        else:

            self._txtInput.setText(self._Beautifier)
        return
```

The content and isrequest parameters are accepted by the isEnabled method; if the isrequest parameter is true, then Burp extender will extract the request using self._extender._helpers.analyzeRequest(content). Accordingly using the request.getHeaders(), we are able to assign the HTTP request headers to the self._headers. With the help of for loop, we can extract the content-type header value and it will match up with the JSON content type list that we defined already. If the HTTP request content type header match with the JSON content type, then the program will fetch the request body by using the [request.getBodyOffset():].tostring(). Afterwards we beautify the request data by using json.dumps() method and json.loads() method.

json.dumps() : we can convert it into a JSON string

json.loads() : we can parse the json data.

We can use the indent keyword argument to specify the indentation size. In the setMessage method, the content will be received and displayed in a new tab. Upon finishing please run the program to see the fruits of your labor.


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action Open Browser
Pretty Raw Actions
JSON Beautifier
Search...
0 matches

Request to https://www.youtube.com:443 [172.217.163.174]

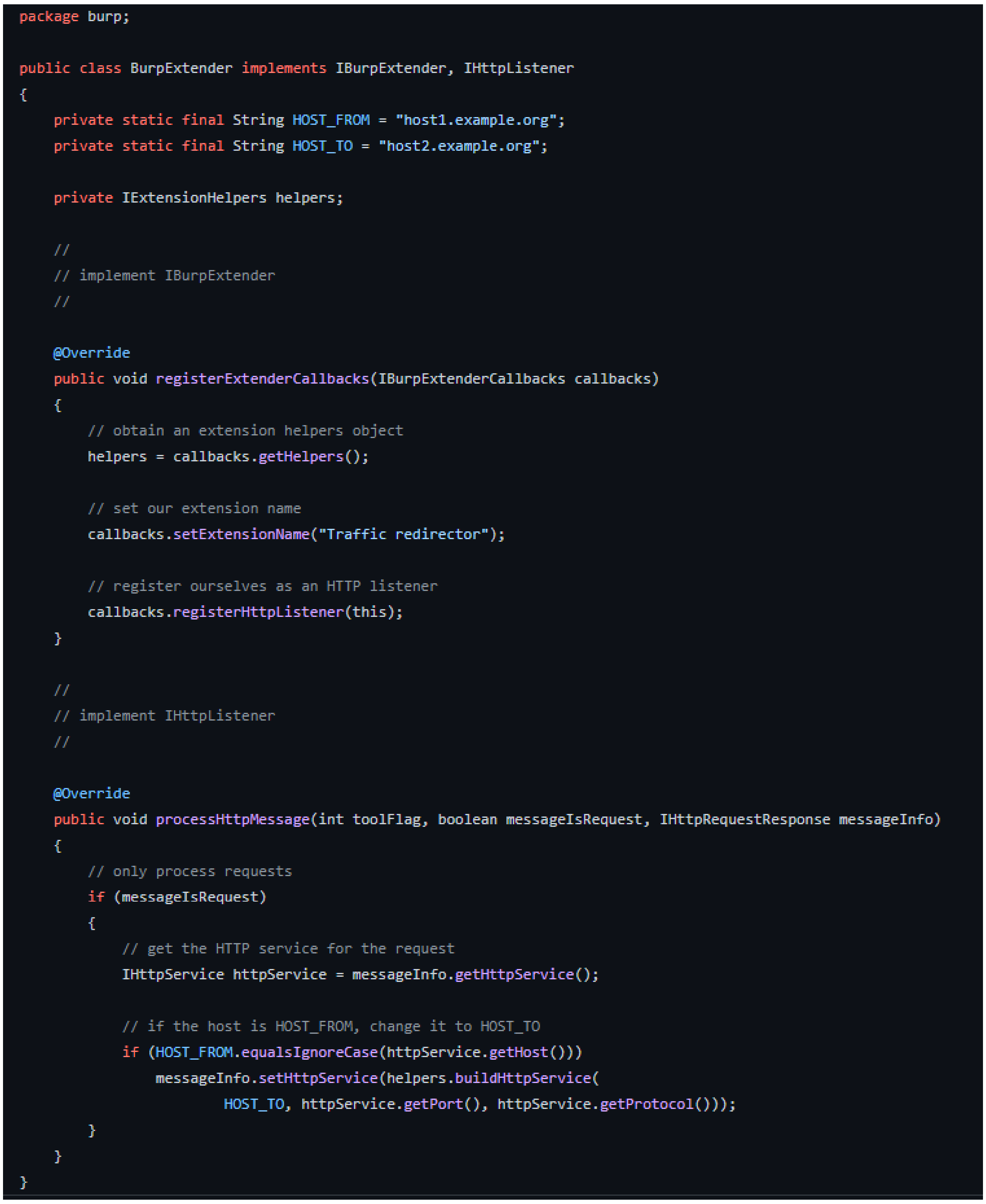
```
                "key": "dt",
                "value": "1616394065918"
            },
            {
                "key": "flash",
                "value": "0"
            },
            {
                "key": "frm",
                "value": "0"
            },
            {
                "key": "u_tz",
                "value": "-240"
            },
            {
                "key": "u_his",
                "value": "3"
            },
            {
                "key": "u_java",
                "value": "false"
            },
            {
                "key": "u_h",
                "value": "1080"
            },
            {
                "key": "u_w",
                "value": "1920"
            },
            {
                "key": "u_ah",
                "value": "1049"
            },
            {
                "key": "u_aw",
                "value": "1920"
            },
            {
                "key": "u_cd",
                "value": "24"
            },
            {
                "key": "u_nplug",
                "value": "3"
            },
            {
                "key": "u_nmime",
```

https://payatu.com/blog/hariprasad/Write-Your-Burp-Suite-Extensions

Sample extensions to get you started

Traffic Redirector in Java

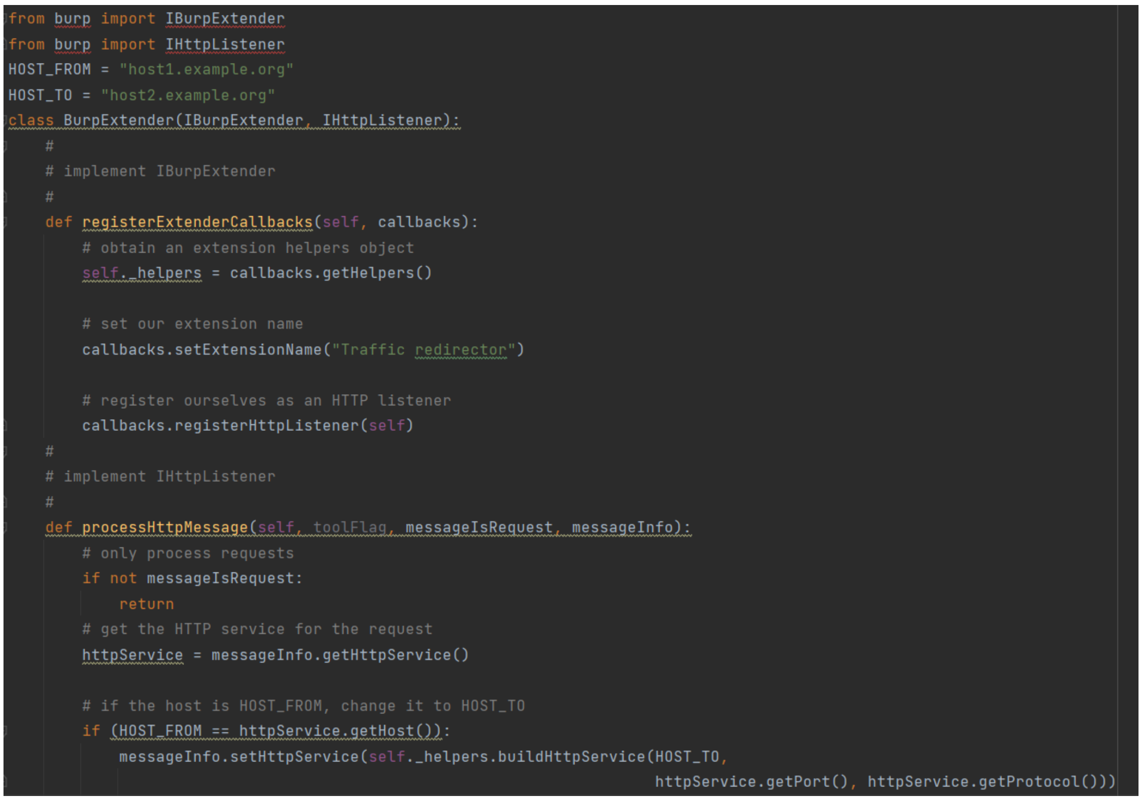
This extension redirects all outbound requests from one host to another.



```
package burp;

public class BurpExtender implements IBurpExtender, IHttpListener
{
    private static final String HOST_FROM = "host1.example.org";
    private static final String HOST_TO = "host2.example.org";

    private IExtensionHelpers helpers;

    //
    // implement IBurpExtender
    //

    @Override
    public void registerExtenderCallbacks(IBurpExtenderCallbacks callbacks)
    {
        // obtain an extension helpers object
        helpers = callbacks.getHelpers();

        // set our extension name
        callbacks.setExtensionName("Traffic redirector");

        // register ourselves as an HTTP listener
        callbacks.registerHttpListener(this);
    }

    //
    // implement IHttpListener
    //

    @Override
    public void processHttpMessage(int toolFlag, boolean messageIsRequest, IHttpRequestResponse messageInfo)
    {
        // only process requests
        if (messageIsRequest)
        {
            // get the HTTP service for the request
            IHttpService httpService = messageInfo.getHttpService();

            // if the host is HOST_FROM, change it to HOST_TO
            if (HOST_FROM.equalsIgnoreCase(httpService.getHost()))
                messageInfo.setHttpService(helpers.buildHttpService(
                        HOST_TO, httpService.getPort(), httpService.getProtocol()));
        }
    }
}
```

Traffic Redirector in Python

This extension redirects all outbound requests from one host to another.



```python
from burp import IBurpExtender
from burp import IHttpListener
HOST_FROM = "host1.example.org"
HOST_TO = "host2.example.org"
class BurpExtender(IBurpExtender, IHttpListener):
    #
    # implement IBurpExtender
    #
    def registerExtenderCallbacks(self, callbacks):
        # obtain an extension helpers object
        self._helpers = callbacks.getHelpers()

        # set our extension name
        callbacks.setExtensionName("Traffic redirector")

        # register ourselves as an HTTP listener
        callbacks.registerHttpListener(self)
    #
    # implement IHttpListener
    #
    def processHttpMessage(self, toolFlag, messageIsRequest, messageInfo):
        # only process requests
        if not messageIsRequest:
            return
        # get the HTTP service for the request
        httpService = messageInfo.getHttpService()

        # if the host is HOST_FROM, change it to HOST_TO
        if (HOST_FROM == httpService.getHost()):
            messageInfo.setHttpService(self._helpers.buildHttpService(HOST_TO,
                                                                       httpService.getPort(), httpService.getProtocol()))
```

Intruder Payloads Java

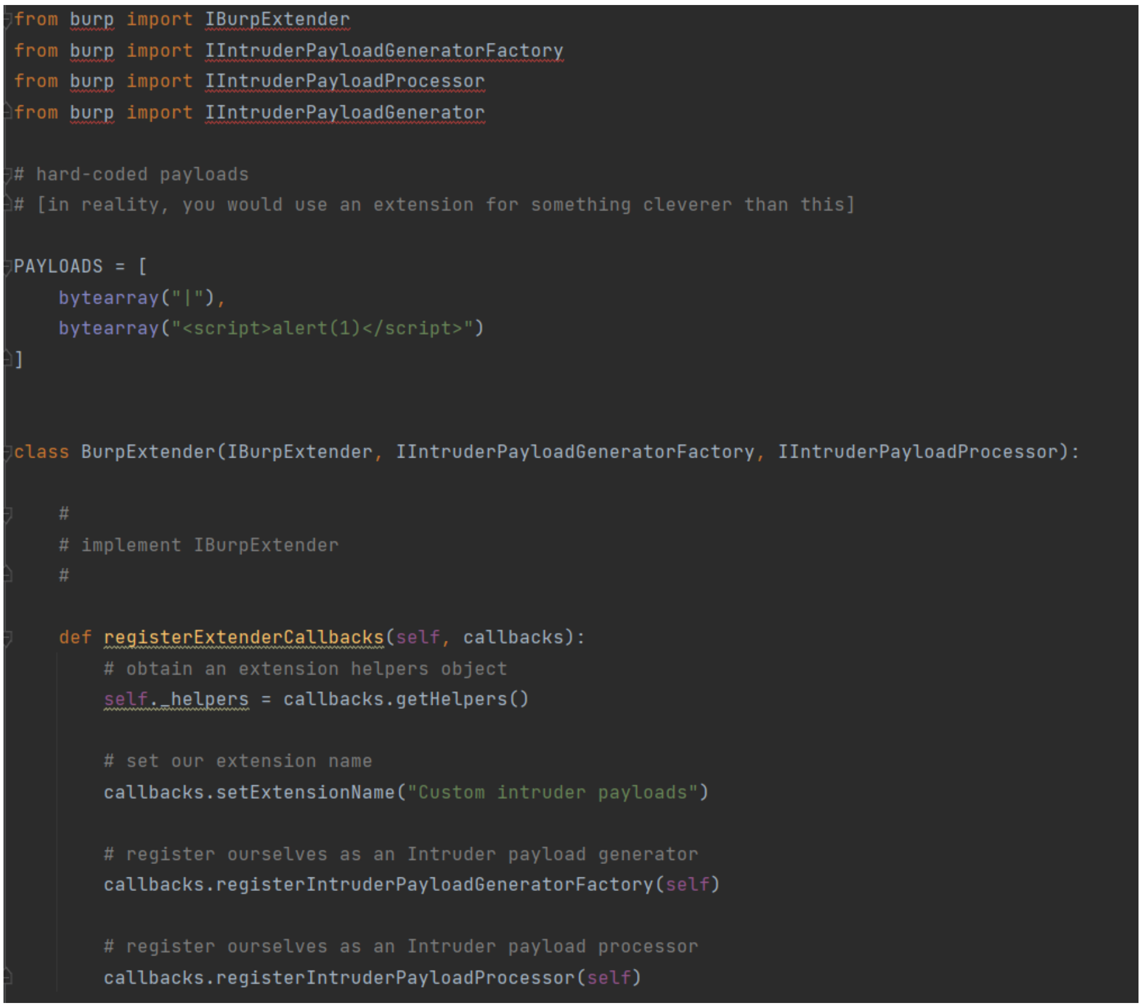
This extension provides custom Intruder payloads and payload processing.

```java
package burp;


public class BurpExtender implements IBurpExtender,
IIntruderPayloadGeneratorFactory, IIntruderPayloadProcessor
{
    private IExtensionHelpers helpers;


    // hard-coded payloads
    // [in reality, you would use an extension for something cleverer than this]
    private static final byte[][] PAYLOADS =
    {
        "|".getBytes(),
        "<script>alert(1)</script>".getBytes(),
    };
```

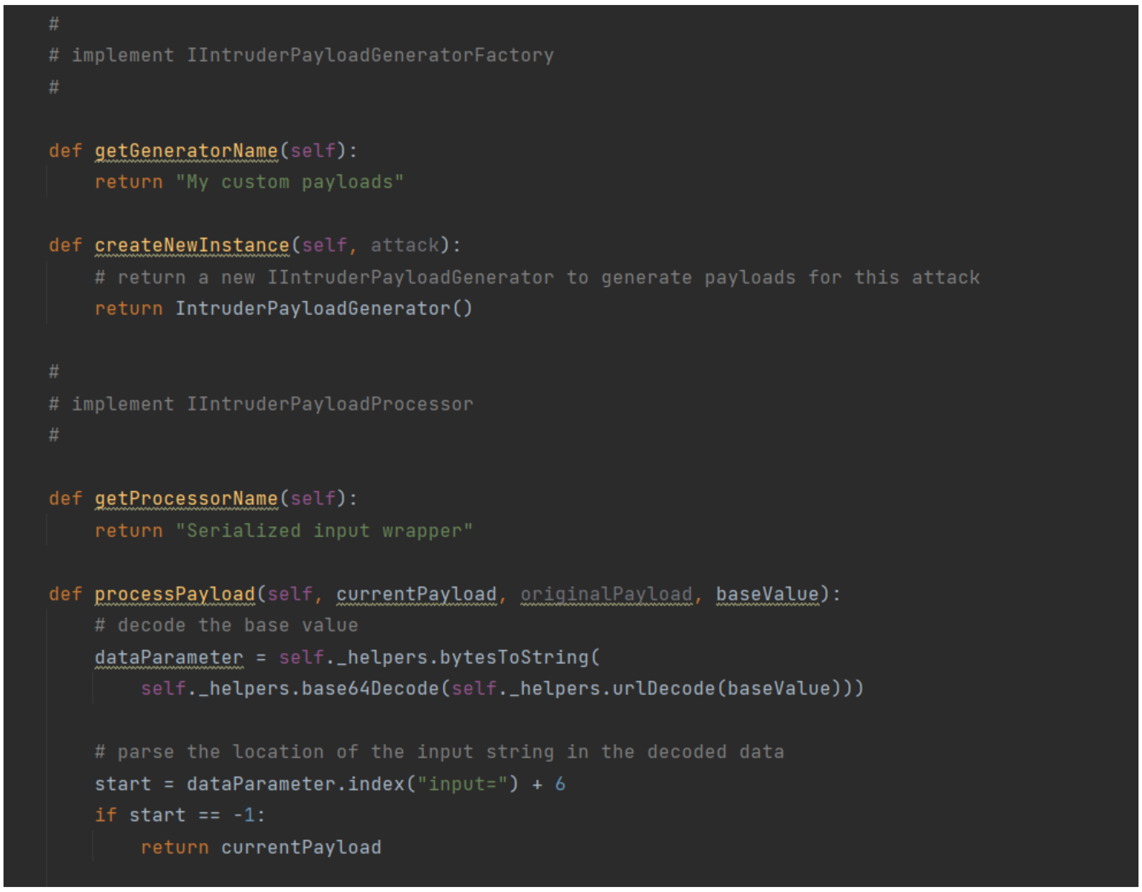
```
//
// implement IBurpExtender
//

@Override
public void registerExtenderCallbacks(final IBurpExtenderCallbacks callbacks)
{
    // obtain an extension helpers object
    helpers = callbacks.getHelpers();

    // set our extension name
    callbacks.setExtensionName("Custom intruder payloads");

    // register ourselves as an Intruder payload generator
    callbacks.registerIntruderPayloadGeneratorFactory(this);

    // register ourselves as an Intruder payload processor
    callbacks.registerIntruderPayloadProcessor(this);
}

//
// implement IIntruderPayloadGeneratorFactory
//

@Override
public String getGeneratorName()
{
    return "My custom payloads";
}
```

```
    @Override
    public IIntruderPayloadGenerator createNewInstance(IIntruderAttack attack)
    {
        // return a new IIntruderPayloadGenerator to generate payloads for this attack
        return new IntruderPayloadGenerator();
    }

    //
    // implement IIntruderPayloadProcessor
    //

    @Override
    public String getProcessorName()
    {
        return "Serialized input wrapper";
    }

    @Override
    public byte[] processPayload(byte[] currentPayload, byte[] originalPayload, byte[]
baseValue)
    {
        // decode the base value
        String dataParameter =
helpers.bytesToString(helpers.base64Decode(helpers.urlDecode(baseValue)));

        // parse the location of the input string in the decoded data
        int start = dataParameter.indexOf("input=") + 6;
        if (start == -1)
            return currentPayload;
        String prefix = dataParameter.substring(0, start);
        int end = dataParameter.indexOf("&", start);
        if (end == -1)
            end = dataParameter.length();
        String suffix = dataParameter.substring(end, dataParameter.length());

        // rebuild the serialized data with the new payload
        dataParameter = prefix + helpers.bytesToString(currentPayload) + suffix;
        return
helpers.stringToBytes(helpers.urlEncode(helpers.base64Encode(dataParameter)));
    }

    //
    // class to generate payloads from a simple list
    //

    class IntruderPayloadGenerator implements IIntruderPayloadGenerator
    {
        int payloadIndex;

        @Override
        public boolean hasMorePayloads()
        {
            return payloadIndex < PAYLOADS.length;
        }

        @Override
        public byte[] getNextPayload(byte[] baseValue)
        {
            byte[] payload = PAYLOADS[payloadIndex];
            payloadIndex++;

            return payload;

        }


        @Override

        public void reset()

        {

            payloadIndex = 0;

        }

    }

}
```

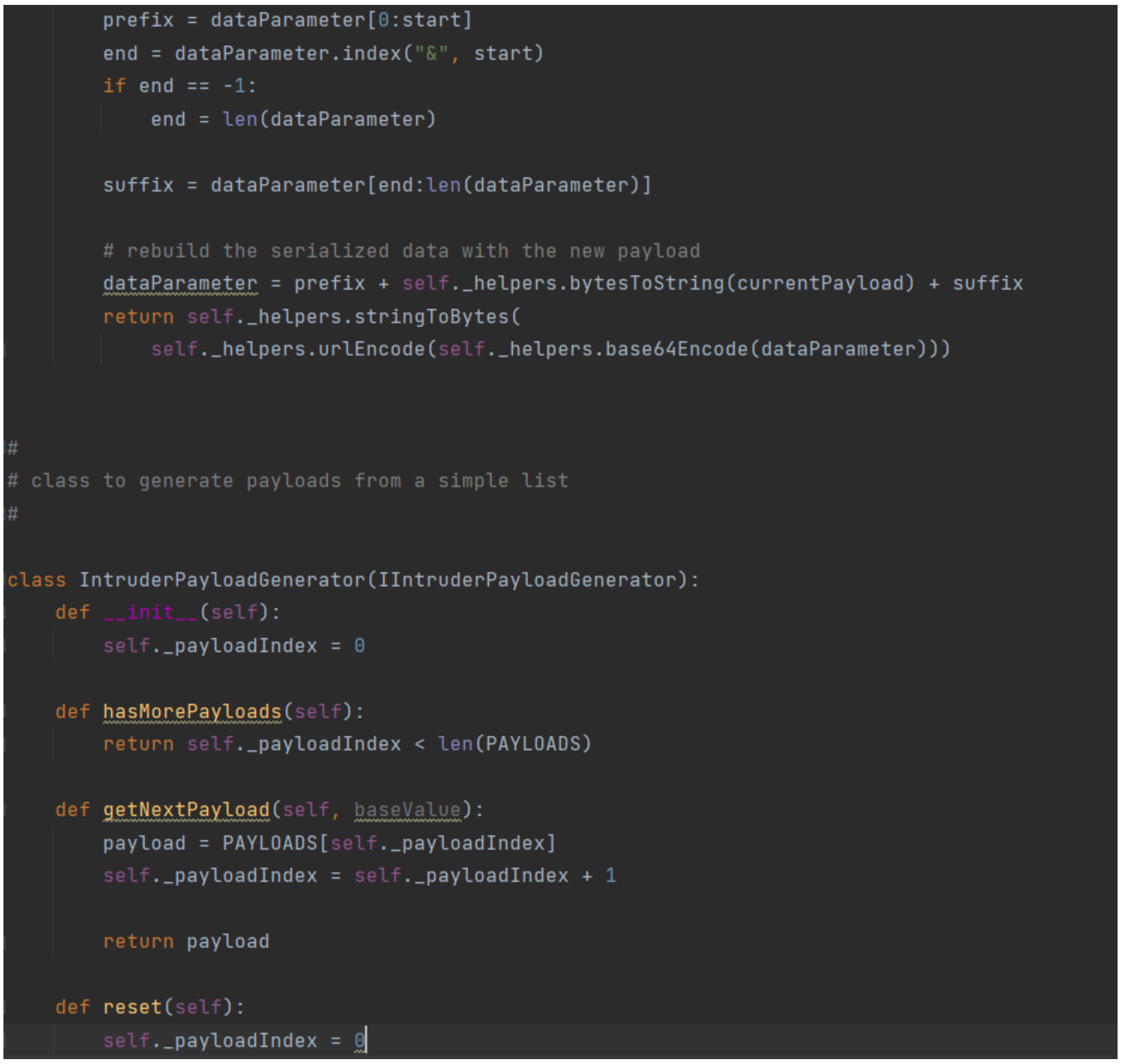
Intruder Payloads in Python



```
from burp import IBurpExtender
from burp import IIntruderPayloadGeneratorFactory
from burp import IIntruderPayloadProcessor
from burp import IIntruderPayloadGenerator

# hard-coded payloads
# [in reality, you would use an extension for something cleverer than this]

PAYLOADS = [
    bytearray("|"),
    bytearray("<script>alert(1)</script>")
]


class BurpExtender(IBurpExtender, IIntruderPayloadGeneratorFactory, IIntruderPayloadProcessor):

    #
    # implement IBurpExtender
    #

    def registerExtenderCallbacks(self, callbacks):
        # obtain an extension helpers object
        self._helpers = callbacks.getHelpers()

        # set our extension name
        callbacks.setExtensionName("Custom intruder payloads")

        # register ourselves as an Intruder payload generator
        callbacks.registerIntruderPayloadGeneratorFactory(self)

        # register ourselves as an Intruder payload processor
        callbacks.registerIntruderPayloadProcessor(self)
```



```
#
# implement IIntruderPayloadGeneratorFactory
#

def getGeneratorName(self):
    return "My custom payloads"

def createNewInstance(self, attack):
    # return a new IIntruderPayloadGenerator to generate payloads for this attack
    return IntruderPayloadGenerator()

#
# implement IIntruderPayloadProcessor
#

def getProcessorName(self):
    return "Serialized input wrapper"

def processPayload(self, currentPayload, originalPayload, baseValue):
    # decode the base value
    dataParameter = self._helpers.bytesToString(
        self._helpers.base64Decode(self._helpers.urlDecode(baseValue)))

    # parse the location of the input string in the decoded data
    start = dataParameter.index("input=") + 6
    if start == -1:
        return currentPayload
```



```
        prefix = dataParameter[0:start]
        end = dataParameter.index("&", start)
        if end == -1:
            end = len(dataParameter)

        suffix = dataParameter[end:len(dataParameter)]

        # rebuild the serialized data with the new payload
        dataParameter = prefix + self._helpers.bytesToString(currentPayload) + suffix
        return self._helpers.stringToBytes(
            self._helpers.urlEncode(self._helpers.base64Encode(dataParameter)))


#
# class to generate payloads from a simple list
#

class IntruderPayloadGenerator(IIntruderPayloadGenerator):
    def __init__(self):
        self._payloadIndex = 0

    def hasMorePayloads(self):
        return self._payloadIndex < len(PAYLOADS)

    def getNextPayload(self, baseValue):
        payload = PAYLOADS[self._payloadIndex]
        self._payloadIndex = self._payloadIndex + 1

        return payload

    def reset(self):
        self._payloadIndex = 0
```

https://portswigger.net/burp/extender#SampleExtensions

Plugin Development Tutorials

https://www.youtube.com/watch?v=IdM4Sc7WVGU

https://www.youtube.com/watch?v=OkQiP_Tcs68

https://www.youtube.com/watch?v=zH0_7Ayfxc4

https://www.vanstechelman.eu/content/creating-and-building-burp-suite-extention-using-java-command-line-tools

https://systemweakness.com/writing-your-first-extension-in-burp-suite-part-1-1645e9bd8f69

https://www.educative.io/courses/burp-suite-extension-development

https://yw9381.github.io/Burp_Suite_Doc_en_us/burp/documentation/desktop/tools/extender.html

https://prakharprasad.com/blog/burp-suite-extension-development-series/

https://devilslab.in/files/Burp_Extension_Writing_Workshop.pdf