渗透-全自动加解密

0x01 前言

在一次项目测试中，发现数据全都是以加密传输并且返回的数据也是全加密，找了一圈发现只能 f0ng 表哥 https://github.com/f0ng/autoDecoder 的工具目前解决问题，但是在测试中发现能够自动解密但是解密的是返回包的数据并且处理数据这里稍微有点问题。我想要的是发给 burp 数据也是解密的内容于是就产生了这款工具 autoDecoder。

0x02 主角

https://github.com/wafinfo/autoDecoder(逆向大佬开发，我负责端菜送水)

0x03 适配环境

可食用各种加解密环境，autoDecoder 只是做个转发功能，加解密功能全在后端 work.py 里面自己实现。

0x04 autoDecoder 食用指南

克隆 https://github.com/wafinfo/autoDecoder 项目下载 jar 包。

Server/demo 目录下有完整的案例如何编写 work.py。平常只用 work.py 文件就行，该文件核心功能主要实现加密。

work.py

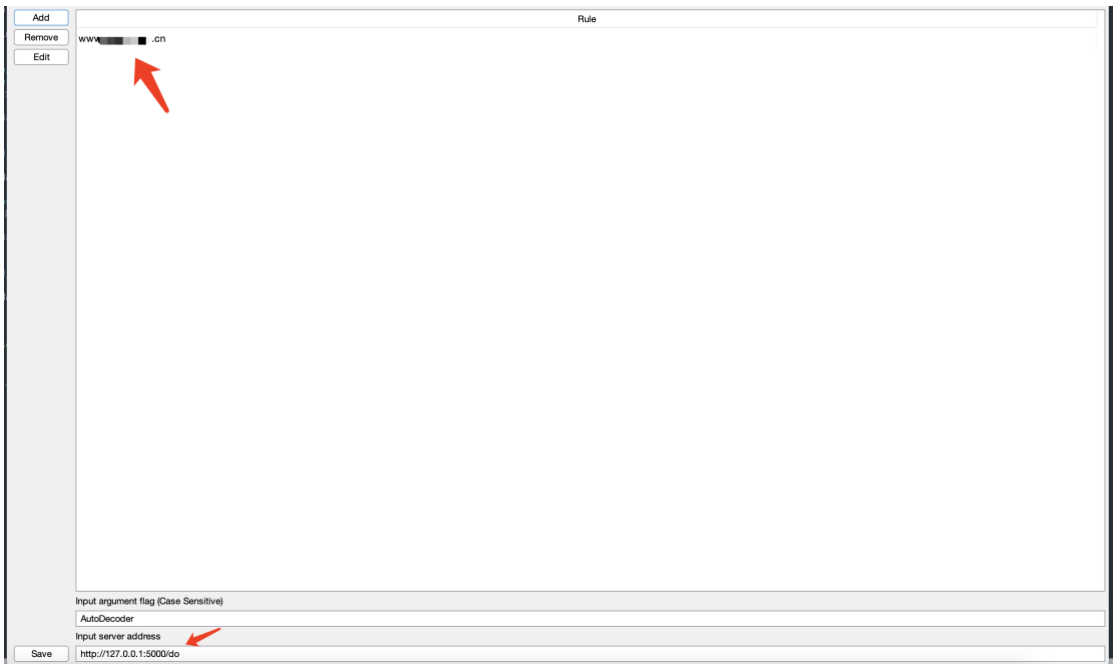
```python

from demo.demo_work import DemoPacket
from framework import BaseModel, Packet, Framework


class Decrypt(BaseModel):
    def on_request(self, data: Packet) -> Dict[str, Any]:
        """处理请求数据

        :param data: Burp中请求数据
        :return: 修改后的数据
        """
        data.add_header("Test", "Response")
        print(f"response:\t{data.body}")
        data.set_body("request=1".encode())
        return data.to_data()

    def on_response(self, data: Packet) -> Dict[str, Any]:
        """处理响应数据

        :param data: Burp中响应数据
        :return: 修改后的数据
        """
        data.add_header("Test", "Response")
        print(f"response:\t{data.body}")
        data.set_body("response=1".encode())
        return data.to_data()


def main():
    Framework("server", Decrypt(), DemoPacket).start()


if __name__ == '__main__':
    main()

```

0x05 小试牛刀

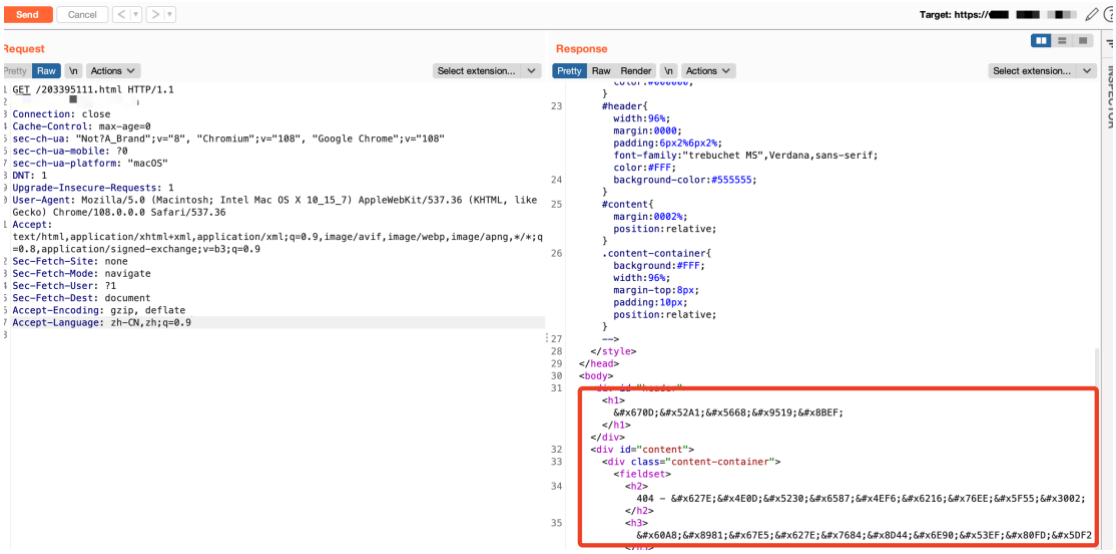
Rule：
配置一个需要加解密的地址通常采用模糊匹配（支持正则）例如：http://xxx.xxx.xxx/abc 只需要 Add 一个 xxx.xxx.xxx 即可。

Flag：
AutoDecoder（可自定义修改只要不与 header 头冲突即可）这个参数标记已经被解密后的数据防止二次解密。例如：当传入进来的数据包是加密状态点击 AutoDecoder 扩展实现解密此时内容已经被解密，进行修改数据的时候标记这个已经是被解密的数据不需要二次解密所以后端判断存在该值就直接进行加密然后发送。

Server：
Work.py 启动的后端实现加解密。



这里配置好后直接启动 demo_work.py 文件。可以看到这里返回的数据是 html 实体编码，修改 work.py 文件。

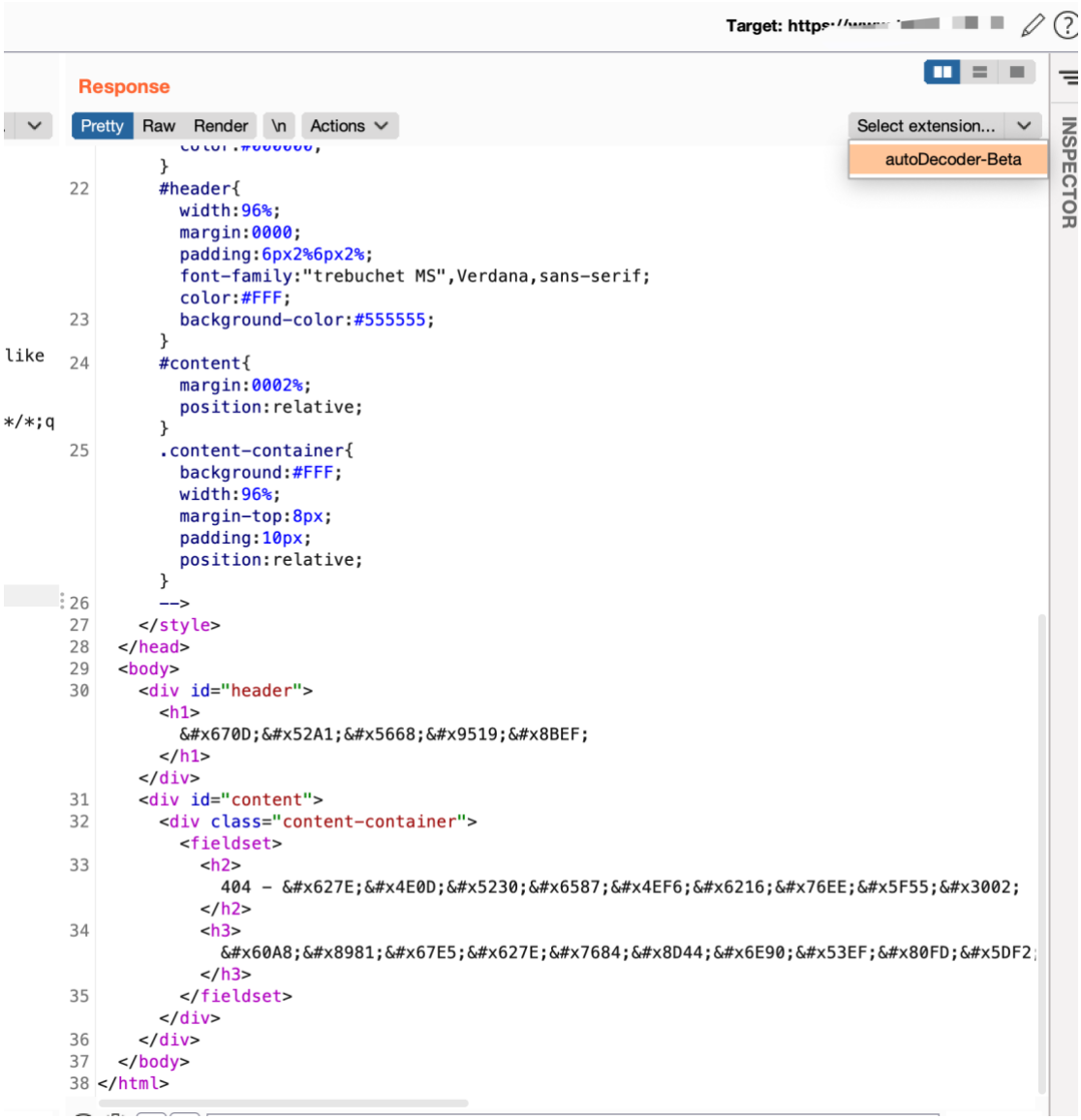


这里由于只有返回包需要进行 html 实体解码所以只需要修改 on_response 方法

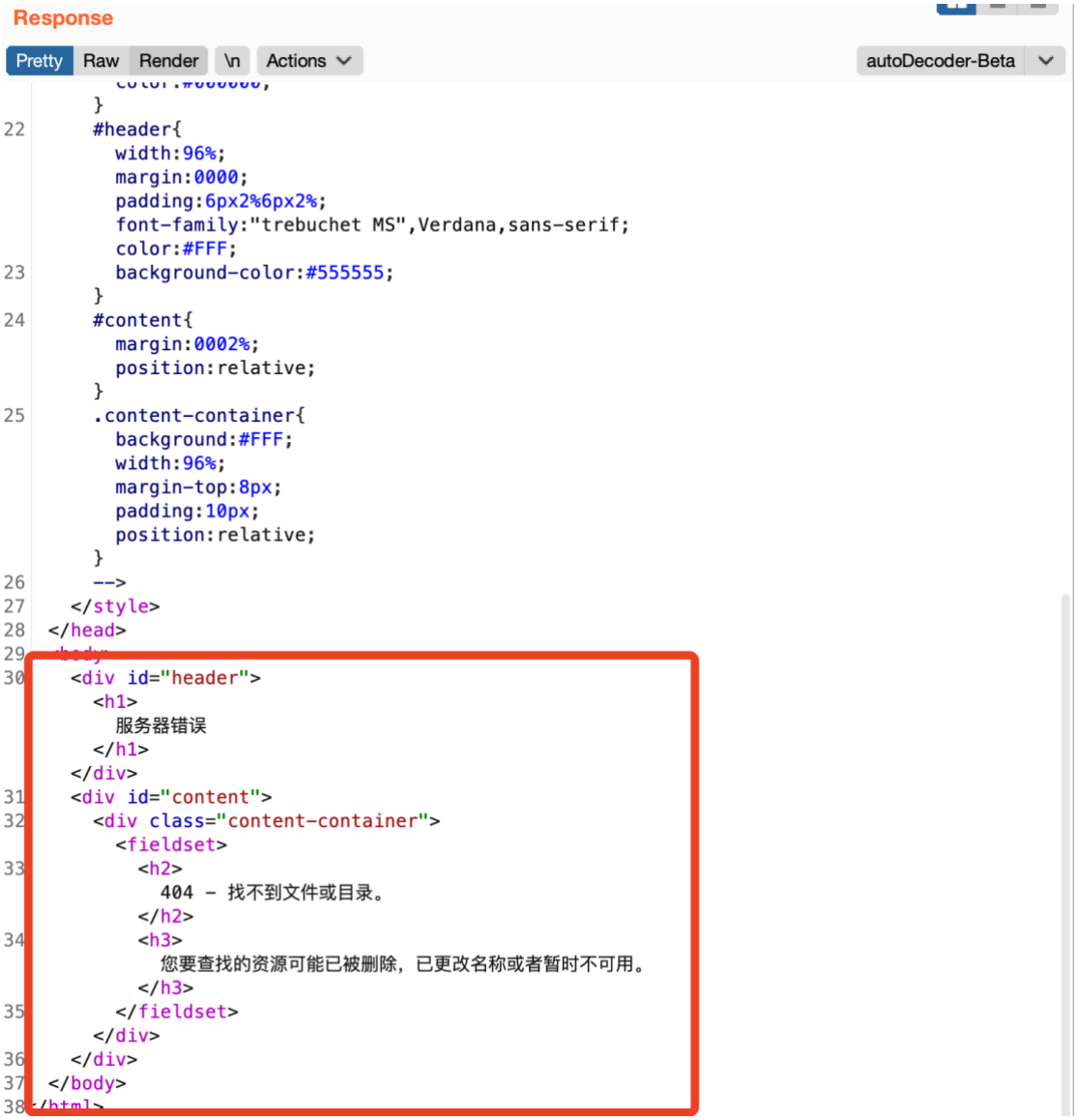
```python
work.py
Decrypt › on_response()
        """处理响应数据

        :param data: Burp中响应数据
        :return: 修改后的数据
        """
        # data.add_header("Test", "Response")
        # print(f"response:\t{data.body}")
        # data.set_body(res)
        data.set_body(html.unescape(data.body.decode()).encode())
        return data.to_data()


def main():
    Framework("server", Decrypt(), DemoPacket).start()


if __name__ == '__main__':
    main()
```

重新启动服务，点击这个插件(我理解这个插件就是刷新内容使用)

```
~/autoDecoder/Server on master +1 !4 ?5 python3 work.py
* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

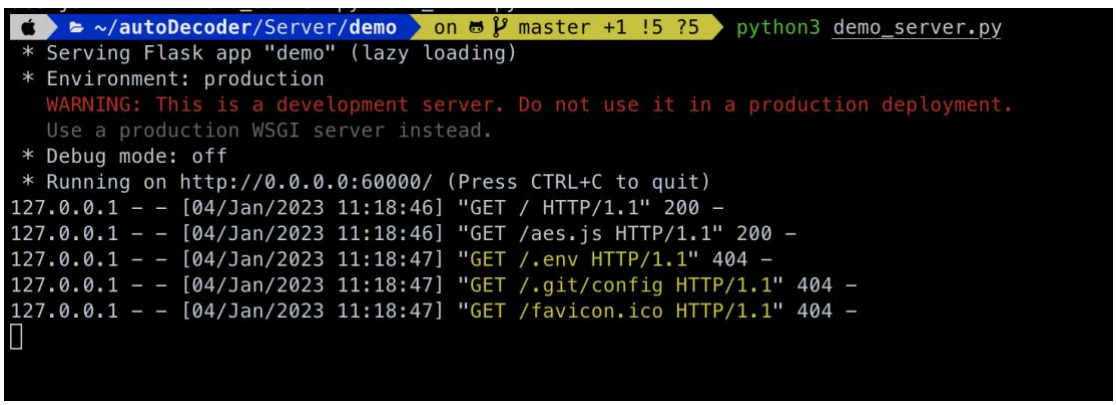


点击 autoDecoder（刷新插件）后切换到 Pertty/Raw/Render 任意模式都可以看到被解码的内容



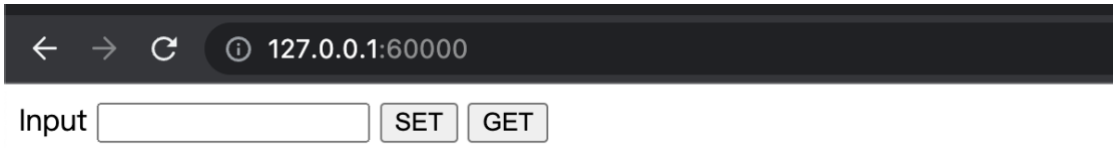
0x06 情景再现

这里模拟了一个当时做的项目环境，环境有点出入但是不影响效果。在 Server/demo 目录下启动 demo_server.py 文件就可以启动模拟的环境。

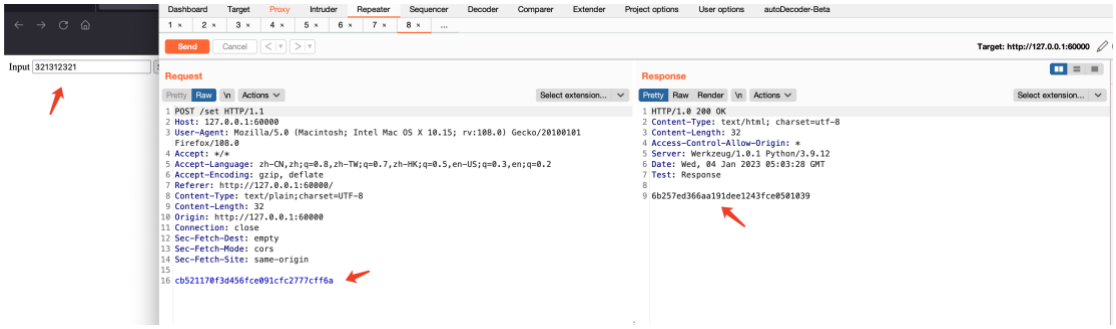


访问地址

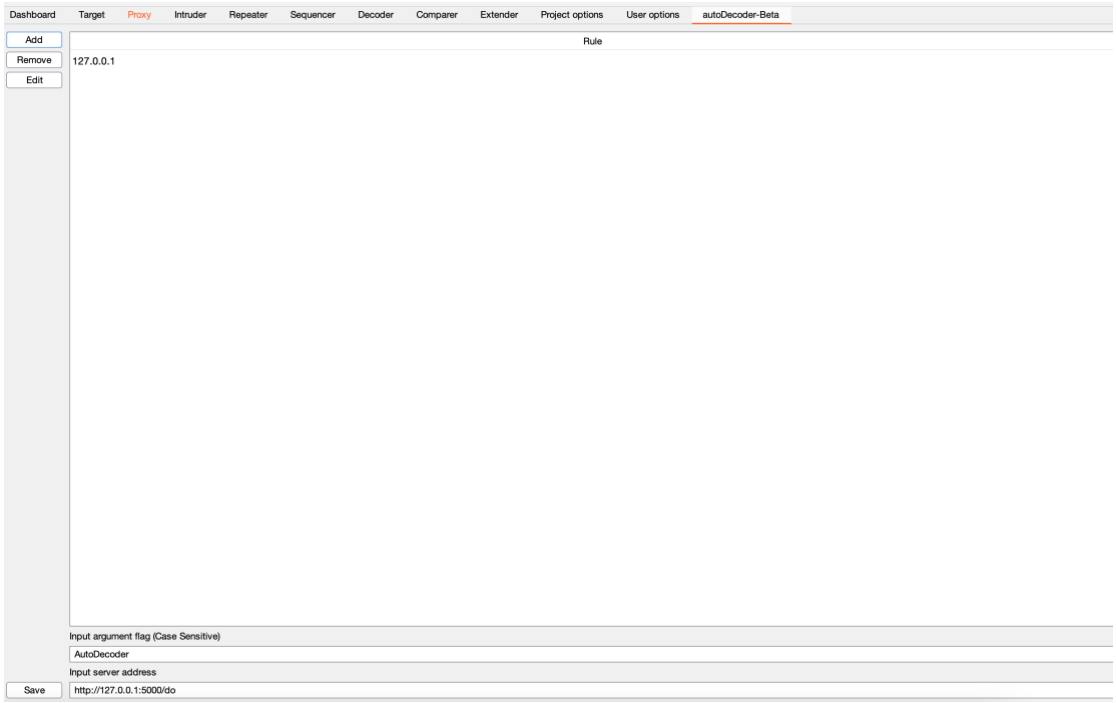
http://127.0.0.1:60000/



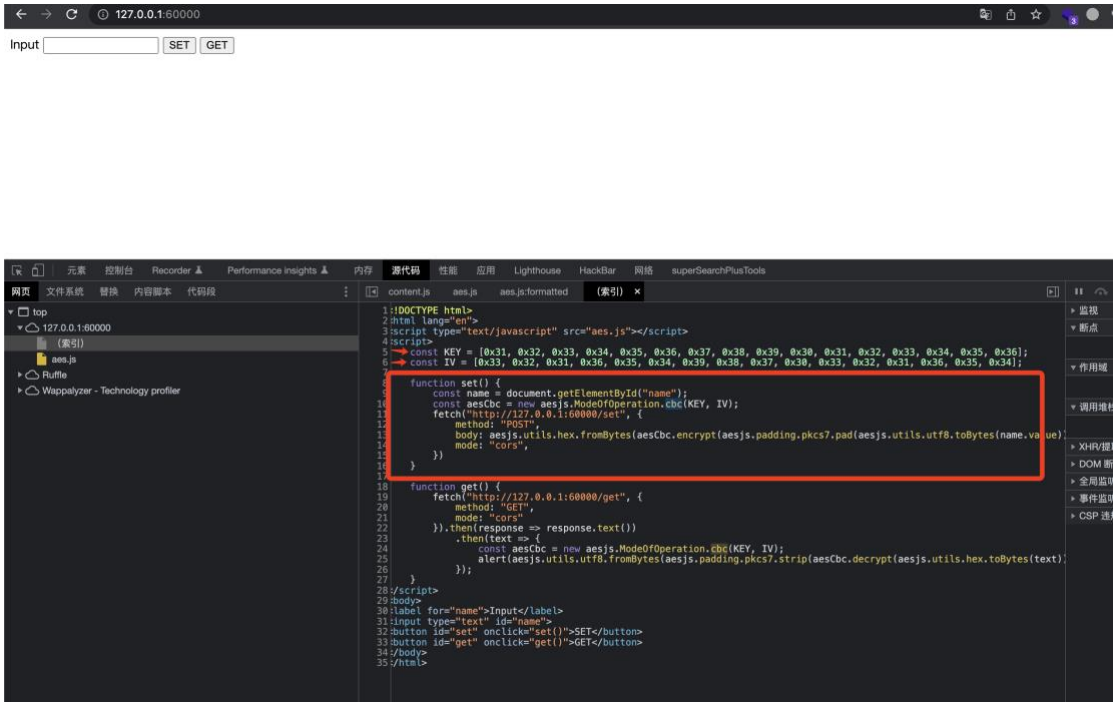
可以看到这里传输是被加密了（项目环境是个 multipart 格式传输）



先添加一个规则，就是被加密的地址。

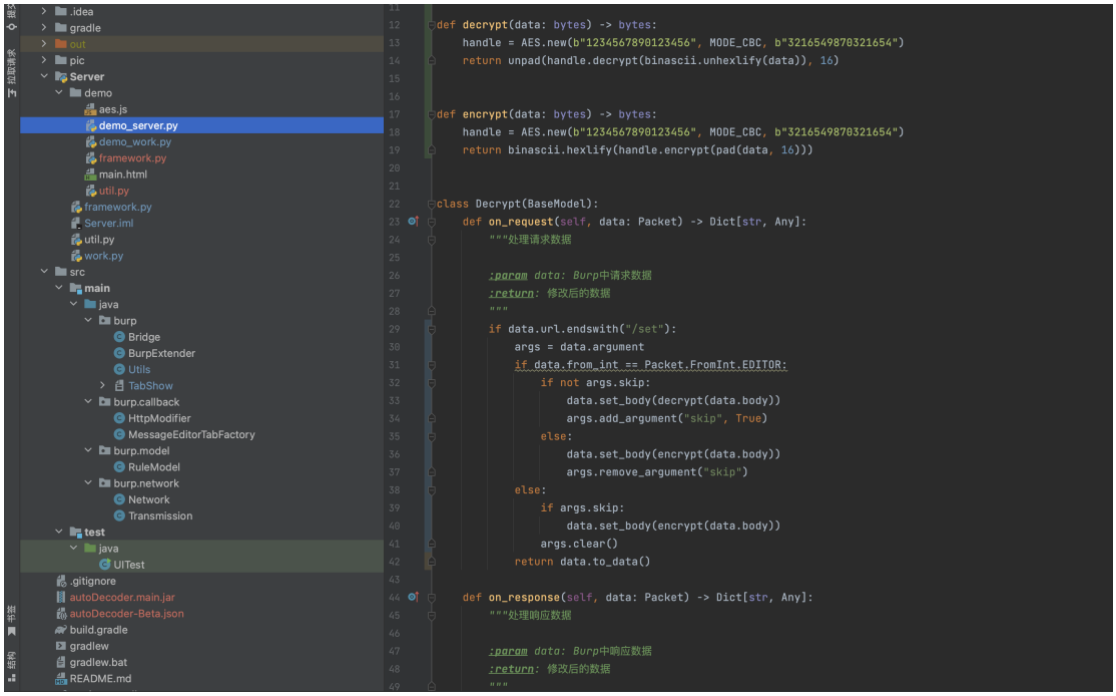


分析这个加密方式是一个 aes 的 cbc 模式（具体实战环境具体分析流程都一样）。



Demo 目录下已经编写好解密 demo_work.py 文件，可直接食用

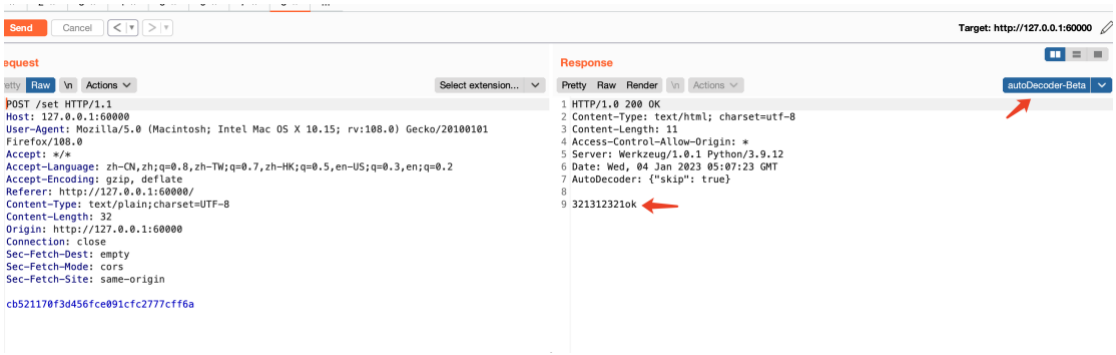
这里还是重新写下 work.py



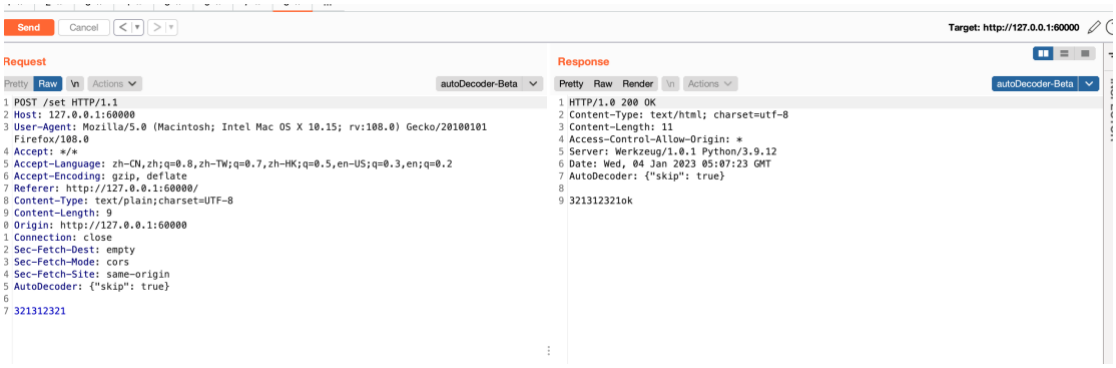
```python
    def on_response(self, data: Packet) -> Dict[str, Any]:
        """处理响应数据

        :param data: Burp中响应数据
        :return: 修改后的数据
        """
        # data.add_header("Test", "Response")
        # print(f"response:\t{data.body}")
        # data.set_body(res)
        if data.from_int == Packet.FromInt.EDITOR:
            args = data.argument
            if not args.skip:
                try:
                    data.set_body(decrypt(data.body))
                    args.add_argument("skip", True)
                except (binascii.Error, ValueError):
                    pass
            else:
                data.set_body(encrypt(data.body))
                args.remove_argument("skip")
        return data.to_data()


def main():
    Framework("server", Decrypt(), DemoPacket).start()


if __name__ == '__main__':
    main()
```

Repeater 模式测试

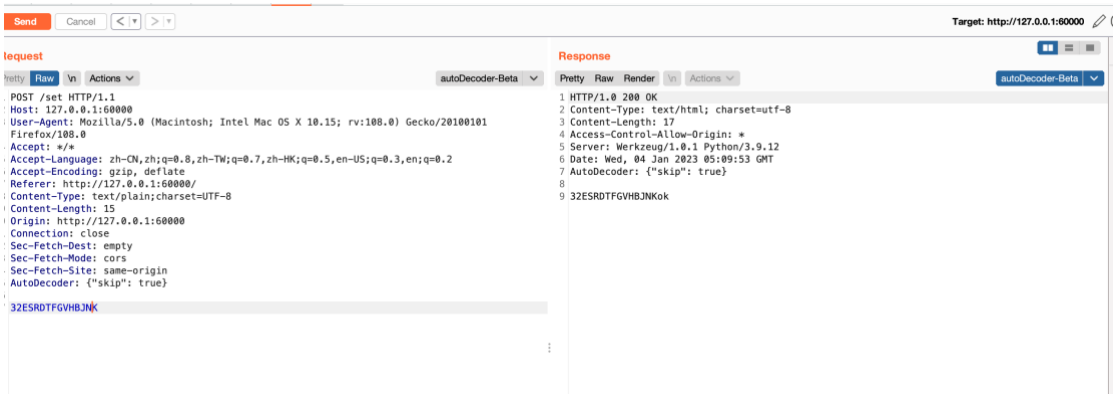
点击 response 的 autoDecoder(刷新插件)返回包就实现了解密



在 request 模式下同样点击实现解密，可以看到请求数据包里面默认自动添加一个头部 AutoDecoder:{"skip":true}（支持自定义在 Ui 界面修改）这个值代表已经解密了防止二次解密，这个参数不会传入到对方服务器，在向服务器发送的时候已经删除了，这个值只是做了判断。



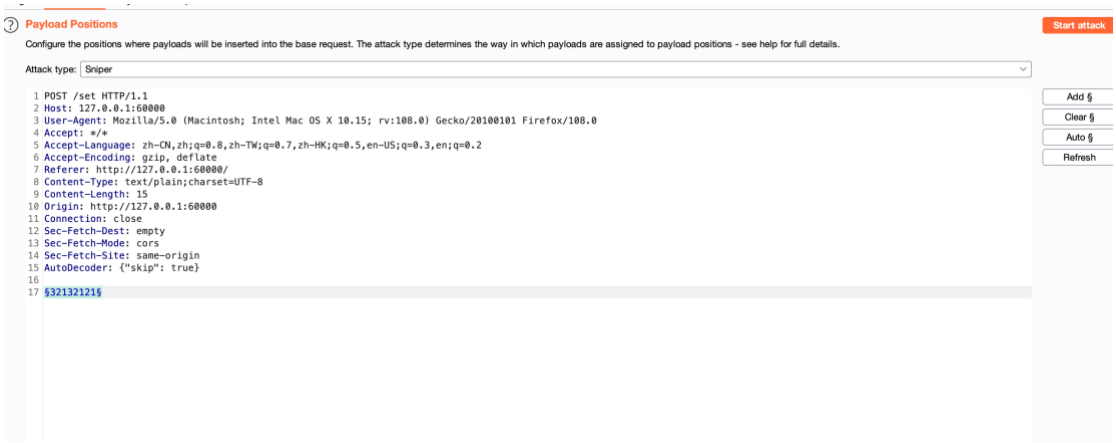
这里进行数据修改，返回包的数据也是也是被修改成功了。

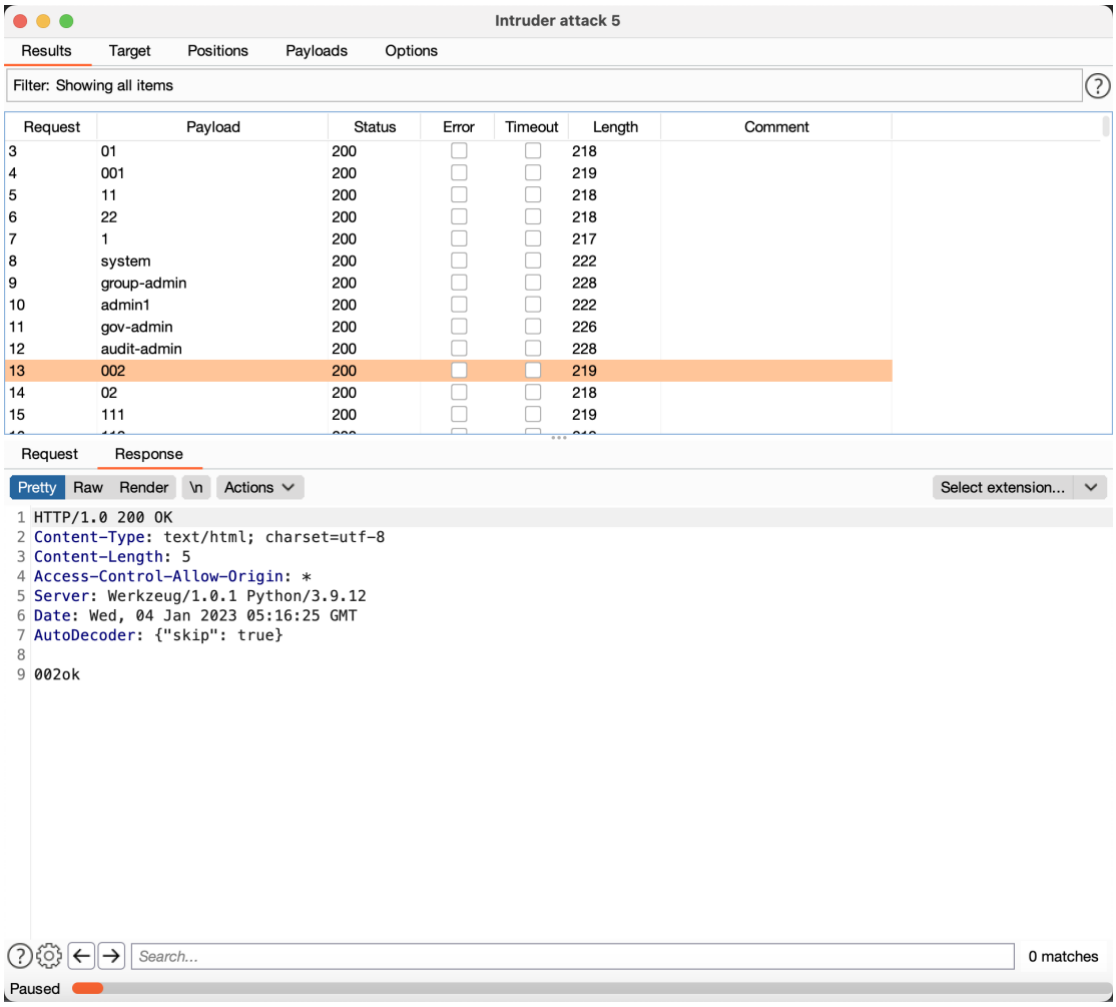


Intruder 模式测试

需要将 EDITOR(编辑模式)改成 TOOL_INTRUDER(爆破模式)，就可以实现返回包自动解密

```python
def on_response(self, data: Packet) -> Dict[str, Any]:
    """处理响应数据

    :param data: Burp中响应数据
    :return: 修改后的数据
    """
    # data.add_header("Test", "Response")
    # print(f"response:\t{data.body}")
    # data.set_body(res)
    if data.from_int == Packet.FromInt.TOOL_INTRUDER:
        args = data.argument
        if not args.skip:
            try:
                data.set_body(decrypt(data.body))
                args.add_argument("skip", True)
            except (binascii.Error, ValueError):
                pass
        else:
            data.set_body(encrypt(data.body))
            args.remove_argument("skip")
    return data.to_data()
```





注意

1、 args = data.argument 是获取之前自定义值也就是判断是否已经被解密了，只要有需要解密的地方一定要写。

2、 Packet.FromInt.EDITOR 是代表模式，常用有 PROXY、INTRUDER、REPEATER、EDITOR 4 种模式，上述也做了案例。

3、 args.remove_argument("skip") 是移除自定义的添加标识符防止发送给服务器。