Adding a complex context-level vulnerability

1. Add vulnerability class.

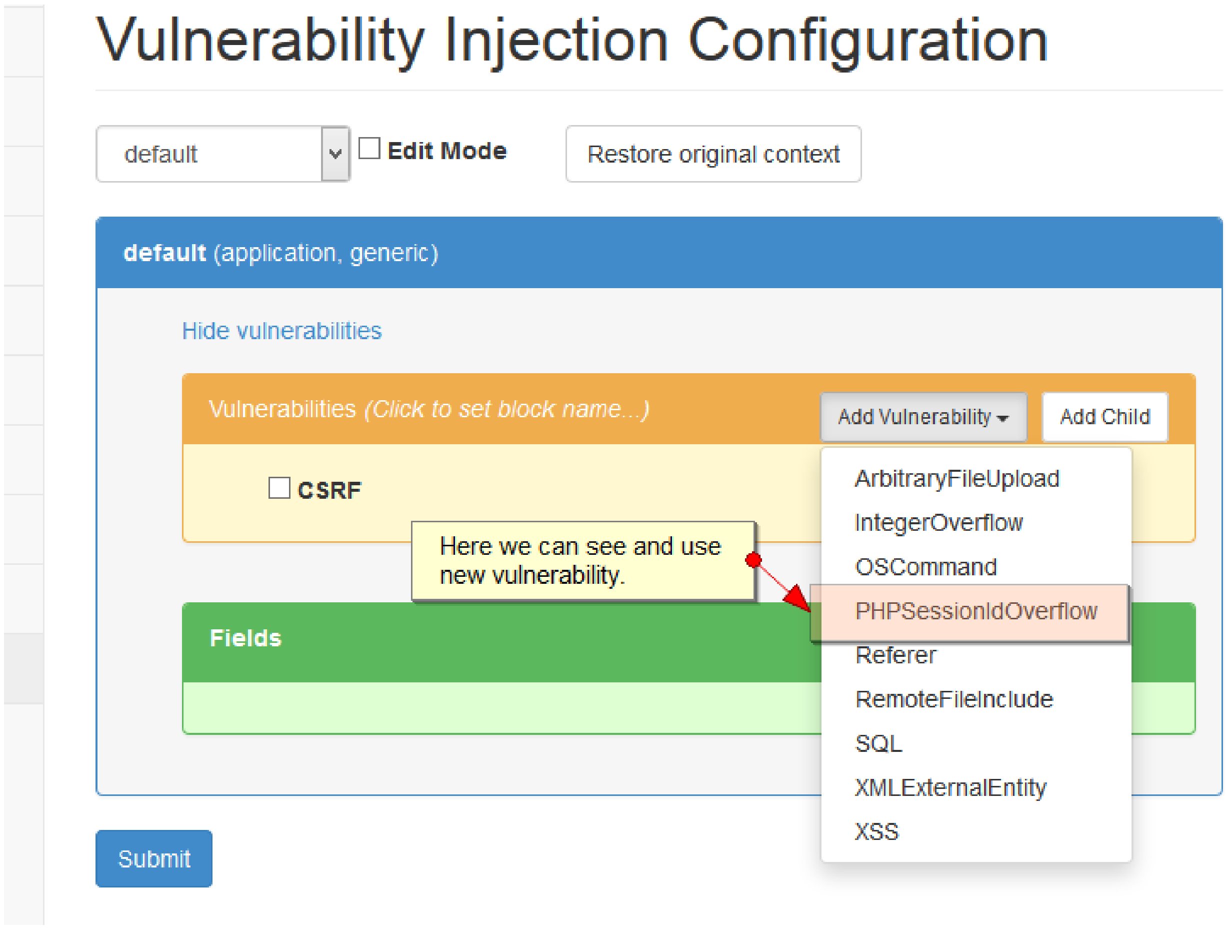
First of all we need to add a vulnerability class itself. Class represents a unique vulnerability with its properties and must extend the VulnModule\Vulnerability class. This class provides basic properties such as whether it's enabled, target entities (context, field), etc. New class must be in the VulnModule\Vulnerability\ directory of the vulnerability module.

For example, we want to add a PHPSessionIdOverflow vulnerability. Add file **`{hackazon_dir}/modules/vulninjection/classes/VulnModule/Vulnerability/PHPSessionIdOverflow.php`** with the following content:

```php
<?php
namespace VulnModule\Vulnerability;

use VulnModule\Vulnerability;

class PHPSessionIdOverflow extends Vulnerability
{
        /**
         * We only use this vulnerability on contexts, not fields.
         * @var array
         */
        protected static $targets = [
                self::TARGET_CONTEXT
        ];

        public function fixSession(){
                // If it's vulnerable, just do nothing.
                if ($this->isEnabled()) {
                        return;
                }

                $sn = session_name();
                if (isset($_GET[$sn]) && strlen($_GET[$sn]) != 32) {
                        unset($_GET[$sn]);
                }
                if (isset($_POST[$sn]) && strlen($_POST[$sn]) != 32) {
                        unset($_POST[$sn]);
                }
                if (isset($_COOKIE[$sn]) && strlen($_COOKIE[$sn]) != 32) {
                        unset($_COOKIE[$sn]);
                }

                if (PHP_SESSION_NONE === session_status()) {
                        session_start();
                    }
            }
    }
```

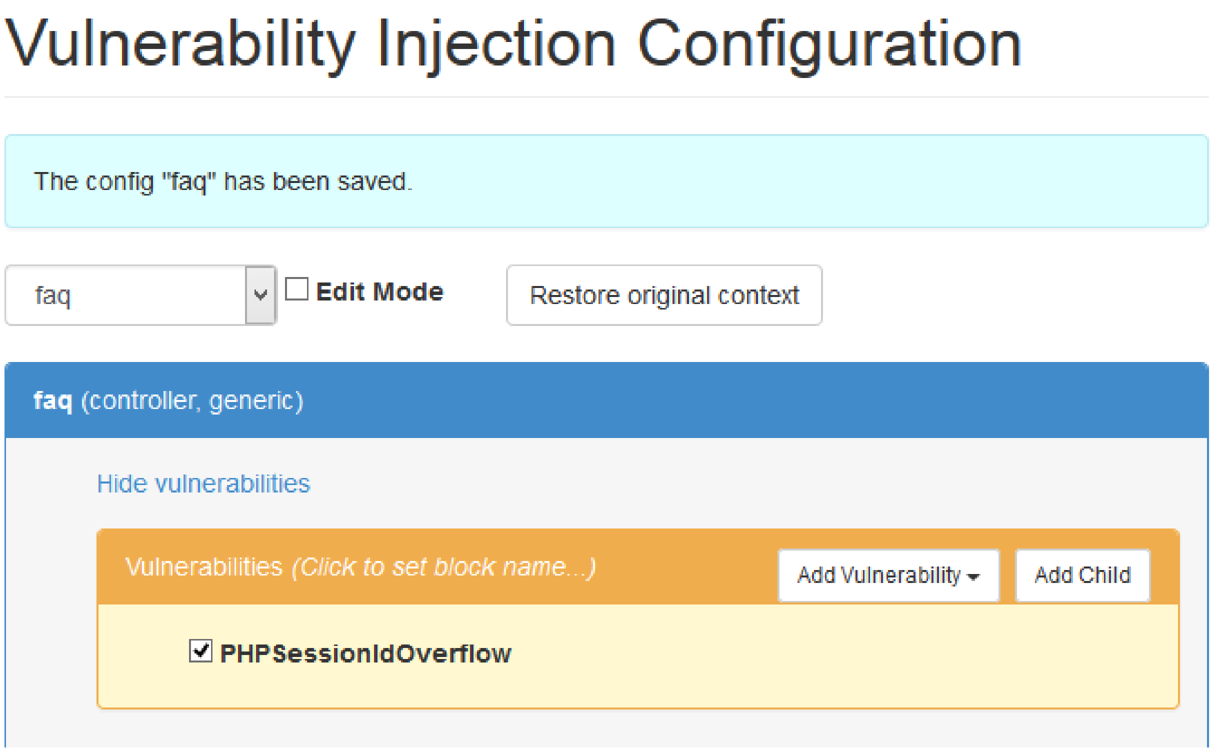
Despite the PHPSESSID value can be treated as a field in cookies, we treat it as a context vulnerability, because the entire context will crash before the moment when we will check it as a cookie field. The whole system depends on this value, not just our app logic. Also, every system can set its own name for a session cookie.

Now we can add this vulnerability:



2. Enable this vulnerability.

Enable vulnerability for some context, for example "faq". By default all vulnerabilities are disabled.


Vulnerability Injection Configuration

The config "faq" has been saved.

faq ☐ Edit Mode

faq (controller, generic)

Vulnerabilities *(Click to set block name...)*

☑ **PHPSessionIdOverflow**

3. Use this vulnerability in the code.

We know that session is not used in controllers until "before()" method, so we add the code to fix session there:

// {hackazon_dir}/classes/App/Core/BaseController.php

```
class BaseController extends Controller
{
        ....
        public function before()
        {
                ...
                $this->vulninjection->goDown($controllerName);

                // Here our code:

                // Get action context
                $actionContext = $this->vulninjection->getCurrentContext()
                        ->getChildByName($this->request->param('action'));

                // Get our vulnerability for this context
```

```
            /** @var PHPSessionIdOverflow $sessVuln */
            $sessVuln = $actionContext
                   ->getVulnerability('PHPSessionIdOverflow');

            // And run the fixing code
            $sessVuln->fixSession();

          // ....
       }
       // ....
}
```

4. Thats it!

Now our vulnerability can be enabled, disabled, or removed. But we may wish to add additional properties to it...

5. Adding additional properties to a vulnerability.

For example we want to be able to control the action that arises when session id value is corrupted but the vulnerability is disabled (e.g. we prevent the buffer overflow). We add a parameter **`$actionOnCorruptedId`** to our vulnerability:

```
class PHPSessionIdOverflow extends Vulnerability
{
       const ACTION_FIX = 'fix';
       const ACTION_EXCEPTION = 'exception';
       const ACTION_HTTP_EXCEPTION = 'http_exception';

       protected $actionOnCorruptedId = self::ACTION_FIX;
       // other code ....

       public function getActionOnCorruptedId()
       {
              return $this->actionOnCorruptedId;
       }

       public function setActionOnCorruptedId($actionOnCorruptedId)
       {
              $this->actionOnCorruptedId = $actionOnCorruptedId;
       }
```

```
	public function asArray()
	{
		return array_merge(parent::asArray(), [
			'on_corrupted_id' => $this->actionOnCorruptedId,
		]);
	}

	public function fillFromArray($data)
	{
		parent::fillFromArray($data);
		if (!is_array($data)) {
			return;
		}

		if (array_key_exists('on_corrupted_id', $data)) {
			$this->setActionOnCorruptedId(
				$data['on_corrupted_id']
			);
		}
	}

	public function equalsTo($vuln)
	{
		if (!parent::equalsTo($vuln)) {
			return false;
		}

		return $this->getActionOnCorruptedId() ===
			$vuln->getActionOnCorruptedId();
	}
}
```

We add property, getter and setter, and override methods to serialize and deserialize vulnerability to config. Also add a equalsTo method for comparisons in vulnerability module core.

OK, we added the property, but how can we edit it in the configuration editor in the admin panel?

We want to add the ability for user to control the behavior of application on corrupted session id when the vulnerability is disabled. It will allow three variants:

- fix the session id (default)
- throw PHP error exception
- throw Http Exception (400 Bad request)

Vulnerability module uses Symfony2 forms component to edit configuration. By default vulnerabilities use a special form type VulnModule\Form\VulnerabilityType. It allows a vulnerability to be visible in editor and user can enable/disable it. But other properties require a separate class which extends VulnerabilityType. Let's add one.

File should be created in directory **`{hackazon_dir}/modules/vulninjection/classes/VulnModule/Form/Vulnerability/`**

Class name should be created using the following pattern: `{VulnerabilityName}Type`.

So add file **`PHPSessionIdOverflowType.php`**:

```
<?php
namespace VulnModule\Form\Vulnerability;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints\Choice;
use VulnModule\Vulnerability\PHPSessionIdOverflow;

class PHPSessionIdOverflowType extends AbstractType
{
        public function buildForm(
                        FormBuilderInterface $builder, array $options)
        {
                $builder->add('actionOnCorruptedId', 'choice', [
                        'label' => 'On corrupted Id',
                        'required' => true,
                        'choices' => PHPSessionIdOverflow::getOnCorruptedActionsLabels(),
                        'constraints' => [
                                new Choice([
                                        'choices' =>
                                                PHPSessionIdOverflow::getOnCorruptedActions()
                                ])
                        ],
                        'attr' => [
                        'class' => 'form-control'
                        ]
                ]);
        }

        public function setDefaultOptions(
                        OptionsResolverInterface $resolver)
        {
                $resolver->setDefaults([
                        'data_class' =>
                        'VulnModule\\Vulnerability\\PHPSessionIdOverflow'
                ]);
        }

        public function getName()
        {
                return 'vuln_php_sess_id_overflow';
        }

        public function getParent()
        {
                return 'vulnerability';
        }
}
```

And register new type in the VulnModule\Form\Extension\VulnExtension extension:

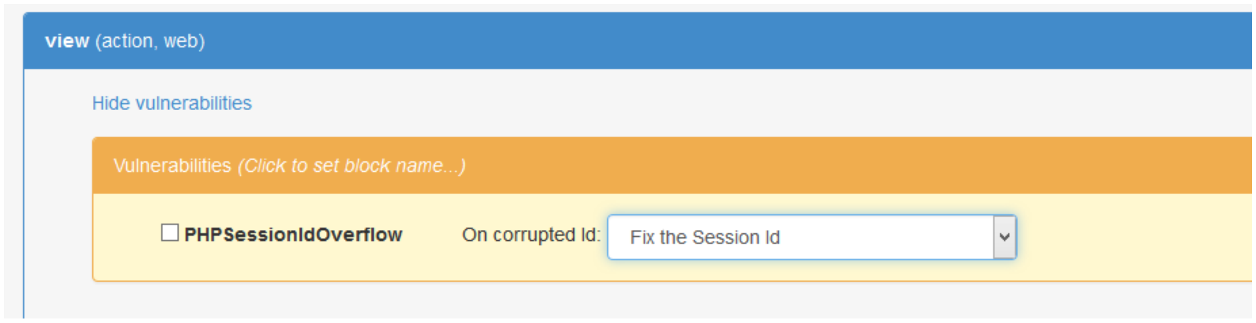
```
class VulnExtension extends AbstractExtension
{
        protected function loadTypes()
        {
                return [
                        // ..............
                        new V\PHPSessionIdOverflowType(),
                        // .............
                ];
        }
}
```

This is a Symfony2 form component's extension which contains all vulnerability module form types.

Also we must add these static methods to our vulnerabilities to be able to get arrays of values for our parameters, and respective labels:

```
class PHPSessionIdOverflow extends Vulnerability
{
        // ..........

        public static function getOnCorruptedActions()
        {
                return [
                        self::ACTION_FIX,
                        self::ACTION_EXCEPTION,
                        self::ACTION_HTTP_EXCEPTION
                ];
        }

        public static function getOnCorruptedActionsLabels()
        {
                return [
                        self::ACTION_FIX => 'Fix the Session Id',
                        self::ACTION_EXCEPTION => 'Throw an Exception',
                        self::ACTION_HTTP_EXCEPTION =>
                                'Throw Http Exception (400 Bad request)'
                ];
        }
}
```

And now we can see that out vulnerability has a select box with values for our parameter:



☐ PHPSessionIdOverflow   On corrupted Id: Fix the Session Id

6. Now that we can edit our parameter, we can use it in our code.

Modify fixSession() method in our vulnerability to utilize new parameter:

```
public function fixSession() {
        if ($this->isEnabled()) {
                return;
        }

        $sessionName = session_name();
        $arrays = [&$_GET, &$_POST, &$_COOKIE];
        foreach ($arrays as $k => $arr) {
                if (isset($arrays[$k][$sessionName])
                        && strlen($arrays[$k][$sessionName]) > 32
                ) {
                        switch ($this->actionOnCorruptedId) {
                                case self::ACTION_EXCEPTION:
                                        throw new \Exception("Invalid session id");
                        case self::ACTION_HTTP_EXCEPTION:
                                unset($arrays[$k][$sessionName]);
                                throw new HttpException(
                                        "Invalid session id. Recreated id.",
                                        400, null, "Bad request");

                        default:
                                unset($arrays[$k][$sessionName]);
                }
            }
        }

        if (PHP_SESSION_NONE === session_status()) {
                session_start();
        }
}
```

Now PHPSessionIdOverflow is fully functional, but one flaw can be fixed: we do not want to see the select box in the config editor unless we disable the vulnerability.

7. Let's add JS features to PHPSessionIdOverflow vulnerability.

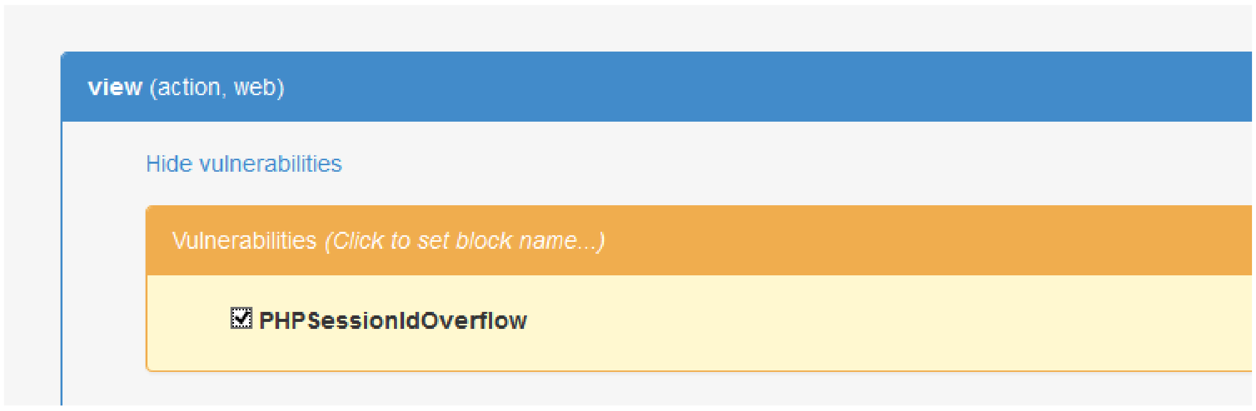
Config uses the CanJS library to dramatically simplify writing jQuery plugins for controls. We add the phpSessionIdOverflowVulnerability plugin which will control the client-side behaviour our vulnerability.
First off, we add some config to form type:

```
class PHPSessionIdOverflowType extends AbstractType
{
        // .............
        public function setDefaultOptions(OptionsResolverInterface $resolver)
        {
                $resolver->setDefaults([
                        'data_class' => 'VulnModule\\Vulnerability\\PHPSessionIdOverflow',
                        'attr' => [
                            'class' => 'form-inline',
                            'data-controller-plugin' => 'phpSessionIdOverflowVulnerability'
                        ]
                ]);
        }

        // ...........
}
```

It will allow the config subsystem to automatically bind the plugin to vulnerability instance. Next, add JS code into **`{hackazon_dir}/web/js/vuln-config.js`**

```
can.Control('PHPSessionIdOverflowVulnerability', {
        pluginName: 'phpSessionIdOverflowVulnerability'
}, {
        init: function () {
                this.attrsElement = this.element.find('.js-vuln-attrs');
                this.enableCheckbox = this.element.find('.js-enable-vuln');
                this.hideOrShowAttrs();
        },

        '.js-enable-vuln change': function () {
                this.hideOrShowAttrs();
        },

        hideOrShowAttrs: function () {
                if (this.enableCheckbox.is(':checked')) {
                        this.hide();
                } else {
                        this.show();
                }
        },

        show: function () {
                this.attrsElement.show();
        },

        hide: function () {
                this.attrsElement.hide();
        }
});
```

Now the select box is hidden when the checkbox is checked:


view (action, web)
Hide vulnerabilities
Vulnerabilities *(Click to set block name...)*
☑ **PHPSessionIdOverflow**

Now vulnerability PHPSessionIdOverflow is completely finished.