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
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
- unimplemented

"sorry but I couldn't make it by the submission deadline :P"
Gyotaku - flag viewer

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

![Screenshot of 192.168.100.1/flag with a forbidden message]

- how to read flag without viewer?
Gyotaku - flag viewer

- /flag is protected with InternalRequiredMiddleware

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

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?
  ○ EICAR signature for testing is enough!

```plaintext
X5O!P%@AP[4\PZX54(P^)7CC)7]$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```
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("EICA"+"R") → detected`
  - argument of `eval` will be audited
- the idea: `eval("EICA"+input) → ?`
  - detected → input is "R"
  - not detected → input is not "R"
Some issues in JScript engine

- **if statement will never be evaluated**
  - if (true) `{eval("EICA" + "R")} → not detected
  - `object accessing` will help you: `{0: "a", 1: "b", ...}[input]

- **parser stops on null byte**
  - `eval("EICA" + "[NULL]") → 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
- \ldots [URL] \ldots [Data] \ldots [UserName] \ldots
  - URL and UserName: controllable
  - Data: secret to be leaked

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

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

```javascript
var body = document.body.innerHTML;
var mal = "EICA";
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

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

```javascript
var body = document.body.innerHTML;
var mal = "EICA";
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
Conclusion

- I presented new Windows side channel 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://github.com/icchy/wctf2019-gtf