HTTP – REQUEST SMUGGLING

PUBLISHED: MARCH 19, 2020 | LAST UPDATE: MAY 12, 2022

SUMMARY

Web application security scans have indicated a potential security weakness when ACOS ADCs are used with some backend web servers. Referred to as HTTP Request Smuggling, this weakness is described in CWE-444 [1] and is addressed in this document.

Researchers have discovered additional methods to perform HTTP Request Smuggling in Web Proxies especially when HTTP/2 is supported at the Proxy for an HTTP/1 backend webserver. This issue is further described in CERT Vulnerability Note VU#357312 [2].

ABOUT HTTP REQUEST SMUGGLING

A deployed ADC configuration, which includes back-end HTTP servers, can be exposed to HTTP request smuggling. CWE-444 provides 2 examples of how this weakness can be exploited. In Example 1, a malformed HTTP request is sent to a website that includes a proxy server and a web server with the intent of poisoning the cache to associate one webpage with another malicious webpage.

In Example 2, a malformed HTTP request is sent to a website that includes a web server with a firewall with the intent of bypassing the web server firewall to smuggle malicious code into the system.

With HTTP/2 enabled or with specially crafted requests that include invalid characters, such as carriage-returns and newlines, requests may be forwarded as HTTP/1 requests that include the malicious data resulting in HTTP Request Smuggling.

AFFECTED RELEASES

This is not a vulnerability in ACOS. A deployed configuration which includes a back-end server and any intermediate HTTP proxying function, can be exposed to HTTP request smuggling. This exposure can be mitigated for ACOS 4.1.1, 4.1.4-GR1, 5.1 and 5.2.x release families as discussed below. Older deployments should upgrade to an ACOS release family that supports mitigations for this weakness.

WORKAROUNDS AND MITIGATIONS

The HTTP request smuggling can be mitigated by enabling the ACOS WAF (Web Application Firewall) feature and adding an ACOS aFlex rule.

Example 1 is mitigated by using the WAF `http-check` or `http-protocol-check` feature which can verify the length information and drops requests with multiple Content-Length headers. The Example 2 is mitigated by using the aFlex rule below to drop POST requests without Content-Type header present.

Although other companies suggest disabling connection-reuse to mitigate HTTP Request Smuggling exposures, A10 Networks' view is that this can help in some cases but is limited in mitigation and will impact performance.

HTTP/2 and invalid character request smuggling mitigations necessitate an additional aFlex script and/or enabling enhanced `http-protocol-check` features available from ACOS release 5.2.1-P4. For the most robust and capable mitigations, consider updating to ACOS 5.2.1-P4 or later.

NOTE: Enabling the ACOS WAF function with default settings could negatively impact the web service being protected. Please refer to the ACOS Web Application Firewall guide to ensure appropriate configuration and settings for the desired operations of the web service.

NOTE: Adding ACOS WAF and aFlex scripts to provide HTTP Request Smuggling mitigations could impact overhead and throughput performance of the ACOS system.

NOTE: Mitigations are indicated below for HTTPS on TCP port 443. They can also be applied for HTTPS on other ports, as well as HTTP on TCP port 80 or other ports.

MITIGATIONS USING ACOS 4.1.1 AND 4.1.4-GR1

For deployments with ACOS 4.1.1 or 4.1.4-GR1 release families, this issue can be mitigated using the following procedure.

1. Add the following aFlex rules.

```
aflex create post-no-content-type
when HTTP_REQUEST {
  if { [HTTP::method] equals "POST" } {
        if { not [HTTP::header exists "Content-Type"] }  {
            HTTP::respond 403 content "<html><head><title>Invalid
request</title></head><body>Invalid request<p></body></html>"
        }
  }
}
.
aflex create multiple-content-length-check
when HTTP_REQUEST {
  if {[HTTP::header exists "Content-Length"] && [HTTP::header exists "Transfer-Encoding"]} {
    reject
  }
}
.
```

2. Configure WAF to perform HTTP checking.

```
waf template wafcheck
  http-check
```

3. Configure the virtual server to use the WAF template and the aFlex rule.

```
slb virtual-server vs-11-1 10.1.11.1
  port 443 https
    aflex post-no-content-type
    aflex multiple-content-length-check
    template waf wafcheck
```

MITIGATIONS USING ACOS 5.1

For deployments with the ACOS 5.1 release family, this issue can be mitigated using the following procedure.

1. Add the following aFlex rule.

```
aflex create post-no-content-type
when HTTP_REQUEST {
  if { [HTTP::method] equals "POST" } {
        if { not [HTTP::header exists "Content-Type"] }  {
            HTTP::respond 403 content "<html><head><title>Invalid
request</title></head><body>Invalid request<p></body></html>"
        }
  }
}
.
```

2. Configure WAF to perform HTTP checking.

```
waf template wafcheck
  http-protocol-check
    multiple-content-length
```

3. Configure the virtual server to use the WAF template and the aFlex rule.

```
slb virtual-server vs-11-1 10.1.11.1
  port 443 https
    aflex post-no-content-type
    template waf wafcheck
```

MITIGATIONS USING ACOS 5.2.X

For deployments with the ACOS 5.2.x release family, this issue can be mitigated using the following procedure.

1. Configure WAF to perform HTTP checking.

```
waf template wafcheck
  http-protocol-check
    post-without-content-type
    body-without-content-type
    multiple-content-length
```

2. Configure the virtual server to use the WAF template

```
slb virtual-server vs-11-1 10.1.11.1
  port 443 https
    template waf wafcheck
```

ADDITIONAL MITIGATIONS USING ACOS 5.2.X AND 4.1.4-GR1

For deployments with ACOS 5.2.1 or 4.1.4-GR1 release families, HTTP Request smuggling exposures related to HTTP/2 and/or invalid characters can be further mitigated using the following procedures.

ACOS 5.2.1-P3/PRIOR AND 4.1.4-GR1

For 5.2.1-P3 and prior, as well as 4.1.4-GR1 releases, include the aFlex script.

1. Also add the following aFlex rule.

```
aflex create SmugglingPrevention_http1and2_521p3_n_prior
when HTTP_REQUEST {
    if { [HTTP::version] equals "2.0" } {
        #If necessary, the following return code and respond string can be customized
        set returnCode 403
        set resp "<html><title>Request Denied!</title><body><center><h1>Request Denied!</h1><p>If you
have any questions contact the admin.</center></body></html>"

        foreach header [HTTP::header names] {

            set value [HTTP::header values $header]
            set count 0

            if { ($value contains "\r") or ($value contains "\n") or ($header matches
{[Tt][Rr][Aa][Nn][Ss][Ff][Ee][Rr][-_][Ee][Nn][Cc][Oo][Dd][Ii][Nn][Gg]*}) or ($header matches
{[Cc][Oo][Nn][Tt][Ee][Nn][Tt][-_][Ll][Ee][Nn][Gg][Tt][Hh]*}) } {
                #Potential Special Character Smuggling or TE/CL header Detected
                HTTP::respond $returnCode content $resp
                HTTP::close
            }

            foreach header1 [HTTP::header names] {
                set value1 [HTTP::header values $header]
                if { ($header equals $header1) and ($value equals $value1) } {
                    set count [expr $count + 1]
                }
            }
            if { ($count >= 2) and ($header matches {[Hh]ost}) } {
                #Duplicated host headers are detected
                HTTP::respond $returnCode content $resp
                HTTP::close
            }
        }
    }
    if { ([HTTP::version] equals "1.0") or ([HTTP::version] equals "1.1") } {
        set count 0
        #If necessary, the following return code and respond string can be customized
        set returnCode 403
        set resp "<html><title>Request Denied!</title><body><center><h1>Request Denied!</h1><p>If you
have any questions contact the admin.</center></body></html>"

        foreach header [HTTP::header names] {

            set value [HTTP::header values $header]
            log "$header"
            if { ($value contains "\r") or ($value contains "\n") or ($header contains "\r") or
($header contains "\n") or ($header contains "\\") or ($header contains "%") or ($header contains "?")
or ($header contains "\(") or ($header contains "\)") or ($header contains "\<") or ($header contains
"\>") or ($header contains "\@") or ($header contains "\,") or ($header contains "\;") or ($header
contains "\<") or ($header contains "\"") or ($header contains "\/") or ($header contains "\[") or
($header contains "\]") or ($header contains "\=") or ($header contains "\{") or ($header contains
"\}") or ($header contains "\t") } {
                HTTP::respond $returnCode content $resp
                HTTP::close
            }

            if { ($header matches {[Tt][Rr][Aa][Nn][Ss][Ff][Ee][Rr][-
_][Ee][Nn][Cc][Oo][Dd][Ii][Nn][Gg]*}) or ($header matches {[Cc][Oo][Nn][Tt][Ee][Nn][Tt][-
_][Ll][Ee][Nn][Gg][Tt][Hh]*}) } {
                set count [expr $count + 1]
            }

            if { ($count >= 2) } {
```

```
                #TE+CL or TE+TE or CL+CL
                HTTP::respond $returnCode content $resp
                HTTP::close
            }
        }
    }
}
.
```

2. Further configure the virtual server to use this aFlex rule.

```
slb virtual-server vs-11-1 10.1.11.1
  port 443 https
    aflex SmugglingPrevention_http1and2_521p3_n_prior
```

ACOS 5.2.1-P4 AND LATER

For 5.2.1-P4 or later releases, include the HTTP SLB template and aFlex script as follows.

1. Add the following slb template.

```
slb template http http-check
    http-protocol-check
        h2up-content-length-alias drop
        malformed-h2up-header-value drop
        malformed-h2up-scheme-value drop
        h2up-with-transfer-encoding drop
        multiple-content-length drop
        multiple-transfer-encoding drop
        transfer-encoding-and-content-length drop
        get-and-payload drop
        h2up-with-host-and-auth drop
```

2. Remove the following WAF template setting to avoid duplication between WAF and HTTP protocol checks.

```
waf template wafcheck
    http-protocol-check
        no multiple-content-length
```

3. Add the following aFlex rules.

```
aflex create SmugglingPrevent_http1and2_521p4_n_later
when HTTP_REQUEST {
    if { [HTTP::version] equals "2.0" } {
        #If necessary, the following return code and respond string can be customized
        set returnCode 403
        set resp "<html><title>Request Denied!</title><body><center><h1>Request Denied!</h1><p>If you
have any questions contact the admin.</center></body></html>"

        foreach header [HTTP::header names] {

            set value [HTTP::header values $header]
            set count 0

            foreach header1 [HTTP::header names] {
                set value1 [HTTP::header values $header]
                if { ($header equals $header1) and ($value equals $value1) } {
                    set count [expr $count + 1]
                }
            }
            if { ($count >= 2) and ($header matches {[Hh]ost}) } {
                #Duplicated host headers are detected
                HTTP::respond $returnCode content $resp
                HTTP::close
            }
        }
    }
    if { ([HTTP::version] equals "1.0") or ([HTTP::version] equals "1.1") } {
        set count 0
        #If necessary, the following return code and respond string can be customized
        set returnCode 403
        set resp "<html><title>Request Denied!</title><body><center><h1>Request Denied!</h1><p>If you
have any questions contact the admin.</center></body></html>"
```

```
        foreach header [HTTP::header names] {

            set value [HTTP::header values $header]

            #Drop http requests with special character in the name or value
            if { ($value contains "\r") or ($value contains "\n") or ($header contains "\r") or
($header contains "\n") or ($header contains "\\") or ($header contains "%") or ($header contains "?")
or ($header contains "\(") or ($header contains "\)") or ($header contains "\<") or ($header contains
"\>") or ($header contains "\@") or ($header contains "\,") or ($header contains "\;") or ($header
contains "\<") or ($header contains "\"") or ($header contains "\/") or ($header contains "\[") or
($header contains "\]") or ($header contains "\=") or ($header contains "\{") or ($header contains
"\}") or ($header contains "\t") } {
                HTTP::respond $returnCode content $resp
                HTTP::close
            }

            #Disable content-encoding: chunked header
            #Do this if necessary
            if { ($header matches {[Cc][Oo][Nn][Tt][Ee][Nn][Tt][-_][Ee][Nn][Cc][Oo][Dd][Ii][Nn][Gg]})
or ($header contains {[Cc][Oo][Nn][Tt][Ee][Nn][Tt][-_][Ee][Nn][Cc][Oo][Dd][Ii][Nn][Gg]}) } {
                if { ($value matches {[Cc][Hh][Uu][Nn][Kk][Ee][Dd]}) } {
                    HTTP::respond $returnCode content $resp
                    HTTP::close
                }
            }

            #Similarly, we could disable custom headers such as Nothing-Interesting if necessary
            if { ($header matches {[Nn][Oo][Tt][Hh][Ii][Nn][Gg][-
_][Ii][Nn][Tt][Ee][Rr][Ee][Ss][Tt][Ii][Nn][Gg]}) } {
                HTTP::respond $returnCode content $resp
                HTTP::close
            }
        }
    }
}
.
```

4. Configure the virtual server to use the slb template and the aFlex rule.

```
slb virtual-server vs-11-1 10.1.11.1
  port 443 https
    aflex SmugglingPrevent_http1and2_521p4_n_later
    template http http-check
    template waf wafcheck
```

SOFTWARE UPDATES

Software updates that address these vulnerabilities are or will be published at the following URL:

http://www.a10networks.com/support/axseries/software-downloads

VULNERABILITY DETAILS

The following table shares brief descriptions for the vulnerabilities addressed in this document.

Vulnerability ID	Description
A10-2020-0001	When malformed or abnormal HTTP requests are interpreted by one or more entities in the data flow between the user and the web server, such as a proxy or firewall, they can be interpreted inconsistently, allowing the attacker to "smuggle" a request to one device without the other device being aware of it.

RELATED LINKS

Ref #	General Link
[1]	CWE-444 Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
[2]	HTTP Request Smuggling in Web Proxies Vulnerability Note VU#357312

ACKNOWLEDGEMENTS

None

MODIFICATION HISTORY

Revision	Date	Description
1.0	2020-03-19	Initial Publication
2.0	2021-07-23	Add mitigation for 5.2.x release families
3.0	2022-05-12	Updates for HTTP/2, invalid character exposures. Includes missing aFlex rule "`multiple-content-length-check`".