Request Mutation and Why WAFs Can't Save You

Dylan Ross

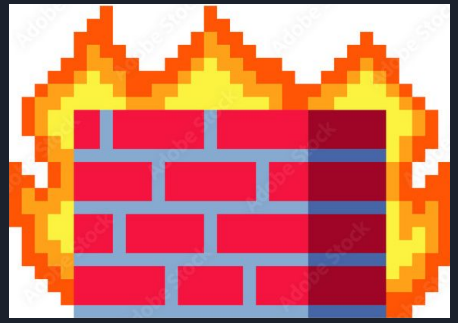
$ whoami

- Pentester at ivision
  - https://research.ivision.com
  - Focus on web apps, APIs, Android stuff, and embedded devices
- RPISEC alumnus
- The guy who factored an RSA key

Request Mutation

It’s not just WAFs

- It is important to note this technique does not just apply to WAFs
  - Input sanitization is another common defense technique that request mutation works against
- Could work for server-side and client-side attacks
- That said, let’s talk about WAFs

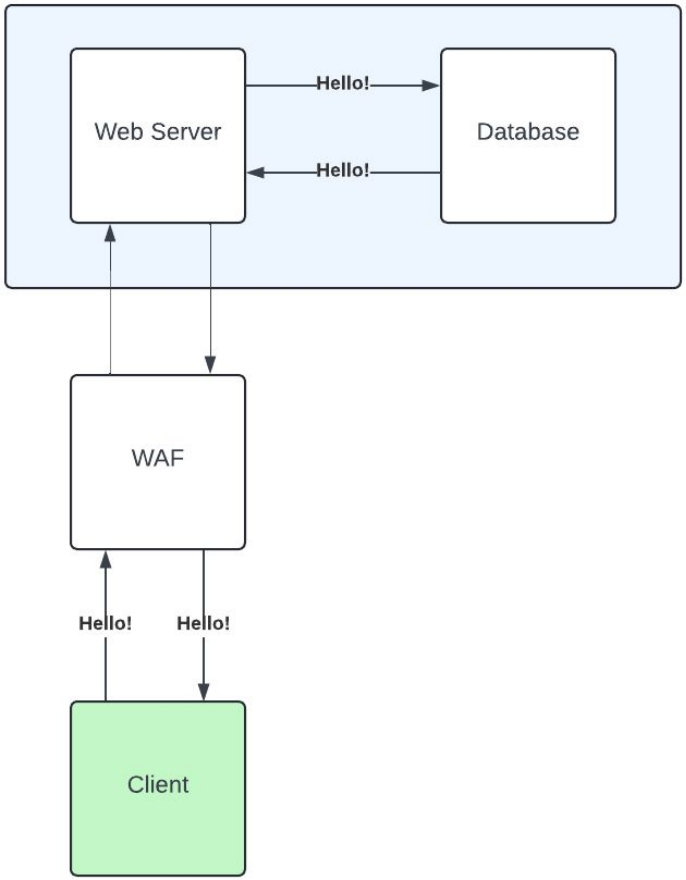


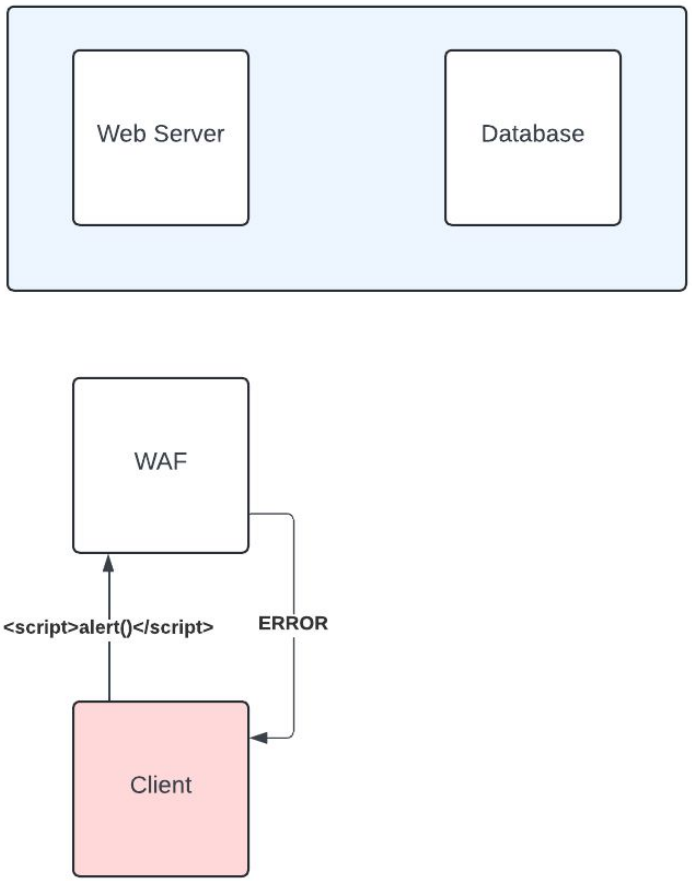
What the WAF?

- Before I go too far, let's talk about what a WAF is
- Web Application Firewall

"A WAF or web application firewall helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. It typically protects web applications from attacks such as cross-site forgery, cross-site-scripting (XSS), file inclusion, and SQL injection, among others." - Cloudflare [1]

[1] https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/


Web Server
Hello!
Database
Hello!
WAF
Hello!
Hello!
Client

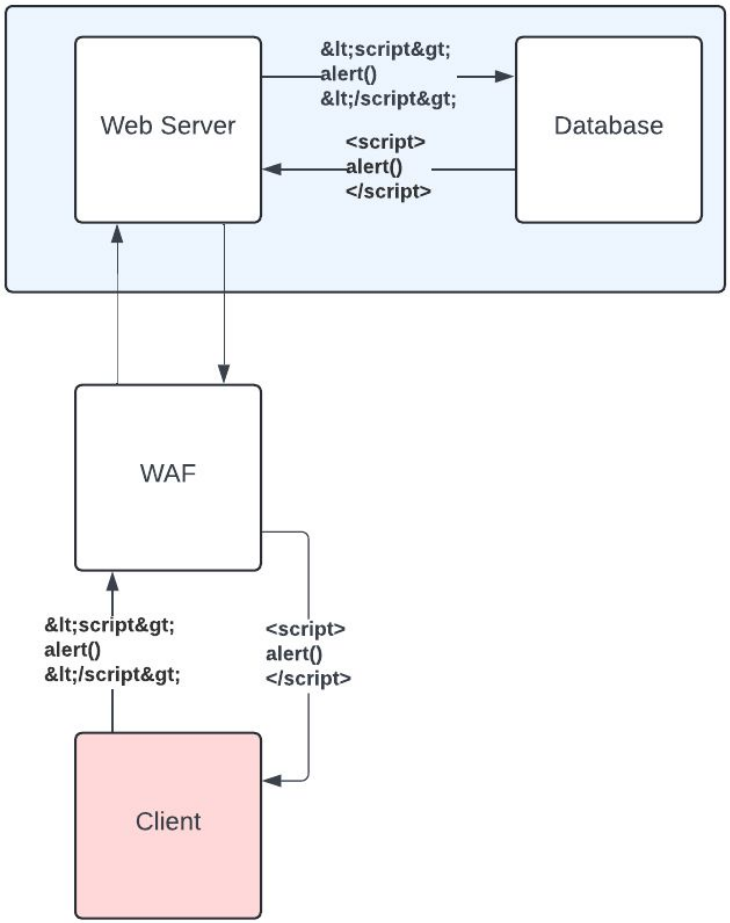

Web Server
Database
WAF
<script>alert()</script>
ERROR
Client

What counts as malicious?

- WAFs need a way to detect malicious requests
- This is done through heuristics and pattern recognition
  - Weird characters like < and >
  - Absolute file paths
  - File uploads that don’t match the declared MIME type

Request Mutation

- Sometimes, the input will change *after* being approved by the WAF
- Imagine a database that HTML-decodes all input
- The WAF no longer knows what's malicious…


Web Server
<script>
alert()
</script>
Database
<script>
alert()
</script>
WAF
<script>
alert()
</script>
<script>
alert()
</script>
Client

Request Mutation

- “A technique where user input is changed ***after*** being approved by a security control into a value that would be blocked by that control” - Me
- WAFs are generic, they can’t know what the server will do to the data after it gets approved
- WAFs, input filtering, etc all work on incoming data. They don’t look at outgoing data

Real Example

Background

- .NET application using the ASP.NET Request Validator
  - Basically a WAF built into the web server
- Microsoft SQL Server
- User input stored in SQL database before being returned in future responses
  - Looking for stored XSS

Request Validator

A potentially dangerous Request.Form value was detected from the client (_ctl1="<script"). - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Media

Address http://localhost/code/ASP.NET1.1/RequestValidation.aspx Go Links

Server Error in '/Code' Application.

A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").

Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").

Source Error:

```
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and
location of the exception can be identified using the exception stack trace below.
```

Stack Trace:

```
[HttpRequestValidationException (0x80004005): A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").]
   System.Web.HttpRequest.ValidateString(String s, String valueName, String collectionName)
   System.Web.HttpRequest.ValidateNameValueCollection(NameValueCollection nvc, String collectionName)
   System.Web.HttpRequest.get_Form()
   System.Web.UI.Page.GetCollectionBasedOnMethod()
   System.Web.UI.Page.DeterminePostBackMode()
   System.Web.UI.Adapters.PageAdapter.DeterminePostBackMode()
   System.Web.UI.Page.ProcessRequestMain()
   System.Web.UI.Page.ProcessRequest()
   System.Web.UI.Page.ProcessRequest(HttpContext context)
   System.Web.CallHandlerExecutionStep.System.Web.HttpApplication+IExecutionStep.Execute()
   System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously)
```

Done Local intranet

Request Validator

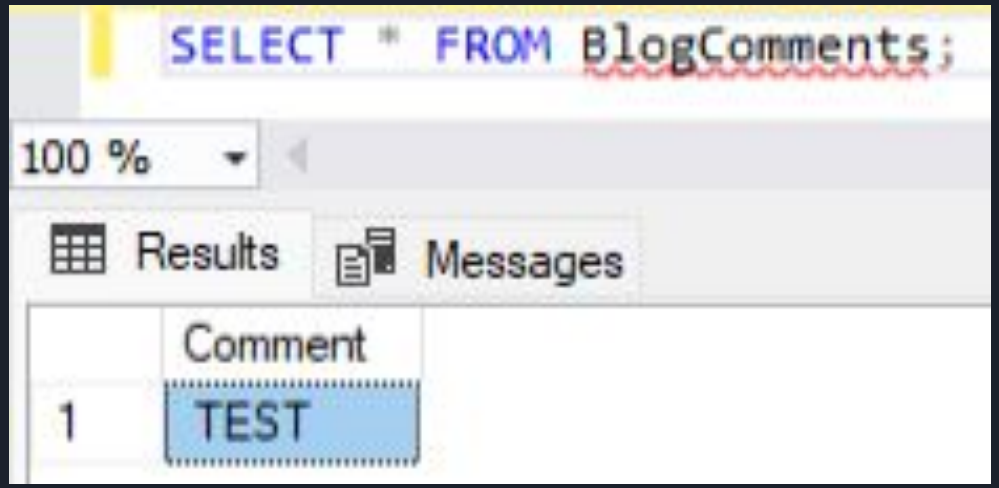
- According to the docs, the request validator looks for unencoded HTML within requests
- Can’t get XSS the old-fashioned way, need to mutate somehow

SQL Server

- SQL Server performs unicode best-fit mapping
  - This isn't very well documented, and it's hard to know that it happens…
- Best-fit mapping effectively takes unicode characters and tries to change them into the ASCII character that it "looks like"
- Can actually see the mappings at http://unicode.org/Public/MAPPINGS/VENDORS/

SQL Server

- Example: test database where the Comment column has type varchar(50)
- Run INSERT INTO BlogComments VALUES ('ŤĘŞŤ');



Best-fit Mapping

- That shows best-fit mapping
- As previously mentioned, the mappings are publicly available

```
0x0118   0x45      ;Latin Capital Letter E With Ogonek
0x015e   0x53      ;Latin Capital Letter S With Cedilla
0x0164   0x54      ;Latin Capital Letter T With Caron
```

Best-fit Mapping

- Maybe we'll find some mappings that are useful to us…

0xff1c 0x3c ;Fullwidth Less-Than Sign

0xff1e 0x3e ;Fullwidth Greater-Than Sign

＜ looks a lot like <, and ＞ looks a lot like >

Now we're getting places

```
INSERT INTO BlogComments VALUES ('<script>alert()</script>');
SELECT * FROM BlogComments;
```

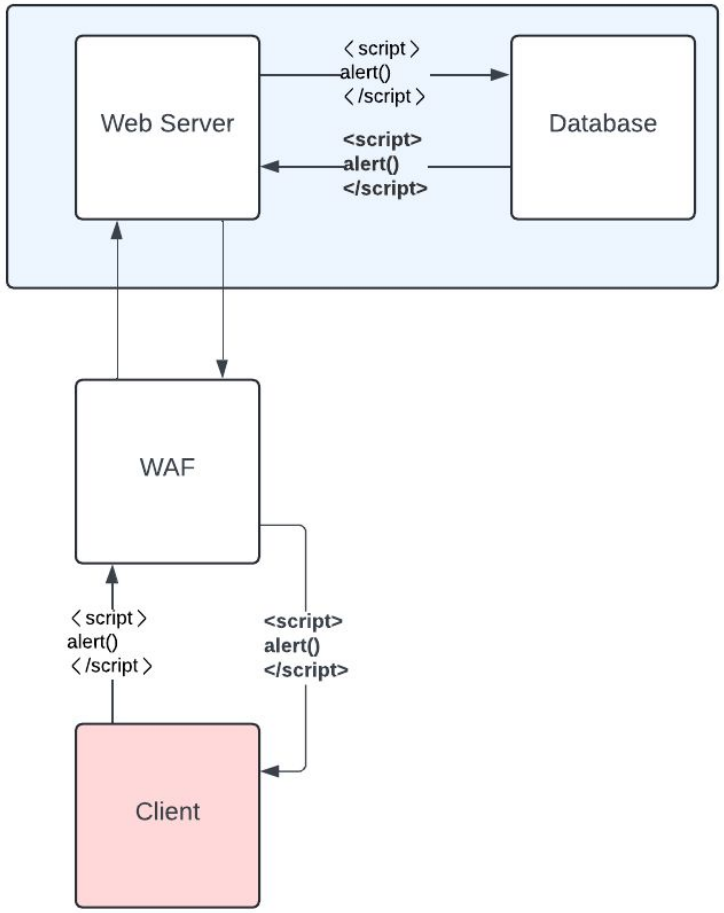
100 %

Results | Messages

	Comment
1	TEST
2	<script>alert()</script>

Request Mutation

- Now we have everything we need for request mutation
  - There are security controls acting on the input that block our attacks
  - The data can be sneakily changed after being checked


Web Server
Database
〈 script 〉
alert()
〈 /script 〉
<script>
alert()
</script>
WAF
〈 script 〉
alert()
〈 /script 〉
<script>
alert()
</script>
Client

Surprisingly Common

- Since I first discovered this behavior, it keeps popping up
  - I now have a favorite Unicode character
- In addition to WAFs, have bypassed custom input filters
- Usually involves Microsoft SQL Server in my experience

Defenses

- WAFs are good as a defense-in-depth
  - They're not sufficient on their own
- Place your security controls as late as possible
  - Location depends on the attack vector
  - Reduce opportunity for mutation after the controls
- Be aware of what software dependencies try to be helpful and change data for you
  - For Microsoft SQL Server, make sure your column can store Unicode
- Use character allowlists where possible

Contact

- Please reach out if you have questions or comments!
- dross@ivision.com