

Forum:[Hack Forums](#)

Thread: [\[TUTORIAL\] Non-conventional WAF/IDS evasion techniques \[PART 3\]](#)

M L T - December 10, 2018, 6:48 am

Non-Conventional WAF/IDS Evasaion techniques - Part 1:

Introduction:

This is the final section of my non-conventional WAF/IDS evasion tutorial series. If you didn't read parts 1 and 2 already, then read those first before reading this thread. They can be found [here](#) and [here](#). At some point in the future I plan to release a traditional WAF evasion tutorial series, which will cover regular filter/signature evasion methodologies unlike this guide which covers non-conventional methods.

Defeating Virtual Patching:

Some poorly-written virtual patches will rely on path data in order to employ their patch. Paths can be manipulated in a manner that they won't match the path specified by the virtual patch, but will still be interpreted as that same path.

For example, take the following mod_security rule for Apache. This is a generic virtual patch which has been applied:

```
Quote:SecRule REQUEST_FILENAME "@streq /path/vuln.php" \
```

```
"chain,phase:2,deny"
```

```
SecRule ARGS:user "!^[a-zA-Z0-9]+$"
```

There are a number of ways in which this path could be accessed in a manner that still loads the intended path, while bypassing the virtual patch in mod_security, for example: /path//vuln.php (additional slashes)

```
//////////path//////////vuln.php?value=PAYLOAD (more slashes - this is fine in PHP)
```

```
/path/./vuln.php?value=PAYLOAD
```

```
/path/blah/./vuln.php?value=PAYLOAD (traversing back a directory + adding arbitrary non-existent directory)
```

```
/path/blah/blah/blah/../../vuln.php?value=PAYLOAD (same as above but traversing multiple times)
```

```
/PaTh/VULN.PHP?VaLuE=PAYLOAD (variations in case-sensitivity)
```

There are some language-specific quirks which can be used to circumvent virtual patching. For example, trailing slashes are fine in PHP but that is not necessarily the case in other languages. The example below is working for Apache Tomcat specifically: /path;/vuln.php?value=PAYLOAD (adding semi-colon)

```
/path;/lol=lol/vuln.php?value=PAYLOAD (adding arbitrary parameter ";lol=lol" and applying to semi-colon)
```

PATH_INFO (the environment variable set via Apache) can also be manipulated in the following fashion: /vuln/vuln.php/lol?value=PAYLOAD (adding arbitrary value to path)

```
/path/vuln.php;lol=lol?value=PAYLOAD (adding arbitrary parameter to path)
```

Just a reminder that these techniques can be used in conjunction with the parameter name

manipulation methods described previously within this guide:
`/path/blah/./vuln.php?+value=PAYLOAD`

This can allow for an extra layer of obfuscation for defeating virtual patching. I'm not going to go in-depth relating to virtual patches here, but feel free to Google the subject or ask any additional questions if necessary. All that you really need to know is that they are sometimes implemented in a fashion that has a hardcoded path which can be bypassed using many of the methods described above.

Session Splicing/Fragmentation:

Splicing or fragmentation of sessions involves splitting your payload into more manageable chunks.

A few techniques already listed in this post attempt to achieve a similar end goal to that of session splicing or session fragmentation. Some examples of this would be, as already mentioned, HTTP Parameter Pollution (wherein the payload is split into chunks and each chunk is 'injected' via the same GET parameter), or HTTP request smuggling.

One well-known tool for performing Session Splicing is called Whisker and it will help automate such a process for an attacker. In effect, what this does is split the payload into multiple smaller packets, allowing them to slip through undetected — meaning that in order for the IDS to detect this, it must reassemble the packet stream. Additional evasion techniques can be implemented by sending the packets out of order and having delays between each transmitted packet. There is a lot of technical information behind this technique, and to explain it in full detail would warrant a blog post of its own. Just remember that this is a potential technique, and that there are tools available to automate such a process. I won't be going into much detail here but look into the Whisker tool if you wish to learn more. Once again, this is bordering on signature-evasion so I will not be going in-depth regarding this topic.

Denial of Service / Resource Exhaustion:

Whereas a WAF or Intrusion Prevention System will block malicious requests, an Intrusion Detection System will simply alert an administrator or any malicious requests. This means an attacker can trigger false positives through means of Denial of Service or other similar techniques, in order to confuse or manipulate the System Administrator — for example, if they get an alert for each attempted DoS attack, alongside an alert for RCE attempts, then an attacker with good timing could trigger a bunch of alerts via DoS, while exploiting the RCE. If all goes to plan, the admin will fail to notice the RCE attempt due to human error, assuming all of the alerts were generated as a result of DoS attempts.

Triggering the likes of Resource Exhaustion or Buffer Overflows within the IDS/IPS or WAF is also a viable and somewhat effective technique.

One method in which resource exhaustion can be achieved is if an attacker takes advantage of the fact that signature-based WAF/IDS software uses pattern matching algorithms to check the signature. Some signatures require more computational power than others to get a match, so, with this in mind, an attacker can send a bunch of specially-crafted packets containing attack signatures that will use up the most CPU power. The end goal here is to cause high load on the CPU, filling up the buffer and causing any new packets sent to be ignored by the IDS due to the fact that the buffer is full.

An attacker can also force the IDS to consume all of its RAM, in which case it reverts to virtual memory on the HDD (drastically slower than RAM) which can result in potentially malicious packets being dropped (read: ignored) by the IDS. This achieves a similar end-result to resource exhaustion through means of an algorithmic complexity attack as described above. Memory exhaustion could be achieved by fragmenting a large amount of packets into an even larger amount of smaller packets, or by initiating a very large number of TCP connections all at once (best to do this in conjunction with out-of-order TCP segments being transmitted).

The purpose of attempting resource exhaustion is to make the IDS ignore any incoming malicious packets due to its buffer being filled up or performance problems created as a result of the IDS reverting from RAM to virtual memory.

That's all for now. In the future, I will be writing a tutorial on traditional WAF bypass techniques using filter evasion to get around signature-based checks, showing you how to build custom payloads which will allow these checks to be circumvented (although be sure to try the techniques listed here before you start wasting time trying to craft a payload ;)