JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

ATTACK SIGNATURE

A security model that can be uniformly applied across all security products, similar to the F5 ASM

Author**: ALI YASLAM OMAR BASALAMA,**

Bachelor of Systems Analysis, Solutions By STC, Jeddah Saudi Arabia, 30 October 2025

1. Attack Signature Definition

Attack Signature is a distinct pattern or identifiable characteristic of a cyberattack that can be recognized by security tools to detect and block malicious activities. These signatures are unique indicators associated with specific attack methods, such as malware, unauthorized access attempts, or network intrusions.

It's essentially a digital fingerprint—a specific pattern, sequence of bytes, or regular expression (RegEx) that correlates exactly to the unique syntax or structure of a known cyberattack or exploit code.

In the realm of cybersecurity, the term "Attack Signature" stands as a sentinel against digital threats. An attack signature, often referred to as a "cybersecurity signature," These patterns can encompass various elements, including the methods employed, malicious code used, and the target's vulnerabilities.

Attack signatures serve as a valuable tool for identifying and mitigating cyber threats, allowing security professionals to stay one step ahead of potential breaches and vulnerabilities.

2. Attack Signatures Pool

This pool acts as the central repository for all known attack patterns. The pool combines:

- System-Supplied Signatures: Signatures created and continually updated by the vendor (F5 in this case) in response to new, widely published vulnerabilities (e.g., those assigned a CVE number).

User-Defined Signatures: Custom patterns created by administrators to address unique application-specific vulnerabilities, proprietary attack methods, or zero-day threats not yet covered by vendor updates.

II. Types of Attacks Detected by Signatures

Attack signatures are categorized to target the specific methods attackers use to manipulate web applications. The broad categories you listed are accurate and are often seen in WAF documentation:

Attack Category	Explanation & Technical Goal of Attacker	Example Payload Signatures
SQL Injection (SQLi)	Attempts to inject malicious SQL code into database queries via user input, aiming to read, modify, or delete data.	Patterns like UNION SELECT, ' OR 1=1 --, or ' AND 1=1.
Cross-Site Scripting (XSS)	Attempts to inject malicious client-side scripts (usually JavaScript) into web pages viewed by other users, typically stealing session cookies or credentials.	Patterns like <script>, onerror=, javascript:alert(, or onload=.
Command Injection	Attempts to execute operating system (OS) commands directly on the server by leveraging vulnerable functions in the application code.	OS commands separated by symbols like `
Directory Traversal (Path Traversal)	Attempts to access files or directories that are outside the web root directory (e.g., configuration files or the /etc/passwd file) using relative path commands.	Patterns like ../, ..\\, or ..\.. combined with common OS files.
Information Leakage	Detects sensitive data being returned in the response (e.g., error messages revealing database syntax, server version numbers, or developer comments).	Messages containing keywords like "ODBC Error", "Warning: mysql_fetch_array()" or version numbers like Apache/2.4.x.
Vulnerability Scan	Detects automated tools (scanners) probing the application for common security flaws (e.g., Nikto, Burp Suite, or Acunetix headers).	Specific HTTP headers, common test payloads, or rapid sequential requests characteristic of scanners.

Key Risks of Attack Signatures

While attack signatures are a crucial part of security defense, they come with inherent limitations and risks:

Zero-Day Threats: Signatures rely on predefined patterns, meaning they are ineffective at detecting zero-day threats (new, previously unknown attacks) that do not yet have associated signatures.

Evasion Techniques: Attackers may bypass signature-based detection systems by obfuscating their code, using encryption, or exploiting polymorphic techniques that modify the signature pattern.

Signature Overload: As attack signatures accumulate, security systems may struggle to manage and process the large number of signatures, potentially leading to false positives or delays in detection.

Delayed Response: There may be a delay between discovering a new threat and releasing an updated signature, leaving systems vulnerable until the update is applied.

Advantages of Using Attack Signatures

Despite limitations, attack signatures offer key benefits as part of a comprehensive security strategy:

Fast Detection of Known Threats: Signature-based detection is highly effective at identifying previously documented threats quickly and accurately, often in real-time.

Low Overhead: Signature-based detection is resource-efficient compared to more complex behavioral or anomaly-based detection methods, not requiring extensive analysis of network traffic or system activity.

Automated Protection: Threat signature systems can operate with little human intervention, providing automated protection by blocking known threats and alerting administrators.

Simpler to Deploy: Signature-based detection systems are generally easier to set up and deploy compared to more sophisticated threat detection methods, making them a popular choice for many organizations.

-Challenges & Considerations

While widely used, there are challenges when implementing threat signature systems:

Dependence on Known Threats: Signature-based systems can only detect threats that are already known and documented, limiting effectiveness against novel or sophisticated attacks without established signatures.

False Positives: These systems may flag legitimate activities as malicious (false positives), leading to unnecessary alerts and potential disruption.

Signature Maintenance: Continuous updates and maintenance of the signature database are necessary to ensure system capability in detecting the latest threats, a time-consuming and ongoing task for security teams.

Evasion and Polymorphism: Advanced attackers can modify malware to evade detection through techniques like encryption, code obfuscation, or polymorphism, altering the attack's appearance without changing its malicious behavior.

Summary: Attack Signatures remain an essential tool for detecting and blocking known security threats. By identifying patterns associated with malicious activity, these signatures provide an efficient and automated way to safeguard against a wide range of threats. However, they are limited by their reliance on known threats and can be circumvented by sophisticated attackers using evasion techniques. For the best security posture, threat signature systems should be used in conjunction with other advanced threat detection methods, such as behavioral analysis and anomaly detection, to ensure comprehensive protection against both known and unknown threats.

The mechanism for updating and enforcing Attack Signatures in a Web Application Firewall (WAF) solution (Example F5 ASM):

The mechanism for updating and enforcing Attack Signatures in a Web Application Firewall (WAF) solution like F5's Application Security Manager (ASM) is based on a three-step technical process: Creation, Distribution, and Inspection.

The fundamental goal is to provide immediate protection against new vulnerabilities (e.g., SQL Injection, Log4Shell) as soon as they are publicly known and assigned a CVE number.

Technical Mechanism Explained

1. Signature Creation (The Research Phase)

When a new vulnerability or attack technique is discovered, the WAF vendor (F5 in this case) rapidly develops a digital fingerprint for it:

Pattern Analysis: Security teams analyze the unique pattern or structure of the malicious code (the payload), often formulating it as a Regular Expression (RegEx). This RegEx is the core of the signature.

Metadata Assignment: This pattern is then packaged with critical metadata:

Signature ID: A unique identifier for the specific attack pattern.

Attack Type: The broad category of the attack (e.g., XSS, Command Injection).

CVE Reference: The public identifier for the vulnerability (e.g., CVE-2021-44228).

Update Period: The timestamp of when this specific signature was created or updated.

2. Distribution and Update (The *.im File)

F5 bundles these new and updated signatures into a compiled binary file (e.g., ASM-AttackSignatures_YYYYMMDD_HHMMSS.im).

Download: The WAF administrator (or a centralized management tool like F5 BIG-IQ) downloads this file from the F5 security update repository.

Installation: The binary file is then installed onto the WAF appliance (the F5 BIG-IP device running the ASM module).

Database Refresh: The installation process loads the new signatures into the WAF's internal signature database, ensuring all security policies instantly benefit from the latest protections.

3. Inspection and Enforcement (Real-Time Traffic)

The WAF applies this updated database to all traffic traversing the system:

Request Interception: Every incoming HTTP Request (and often the outgoing Response) is intercepted by the WAF engine.

Deep Inspection: The engine analyzes key components of the request, including the URL, headers, parameters, and the body of the message.

Pattern Matching: The engine performs rapid comparisons of the request content against the thousands of active RegEx patterns stored in the updated signature database.

Action:

If no match is found, the request is allowed to proceed to the application server.

If a match is found (meaning a signature is present in the request), the WAF considers it a violation.

Blocking: The request is immediately rejected.

Logging: A security alert is logged with the label "Violation attack signature detected" and the request is terminated, typically returning an HTTP response code to the client (often a 403 Forbidden).

In summary, the technical process ensures that the WAF's rule set remains a current, digital fingerprint library capable of identifying and blocking known attacks before they ever reach the application code.

Attack signatures, or threat signatures, are used by many cybersecurity services beyond F5's Application Security Manager (ASM) to detect known threats

These signatures are unique identifiers, such as a file hash, a specific sequence of bytes, or a known pattern of network activity, that identify malicious code or behavior.

Here are other security services that use attack signatures:

-Intrusion Detection and Prevention Systems (IDPS)

Signature-based IDPS: Many IDPS use a database of predefined attack signatures to monitor network traffic.

Function: When a packet or log entry matches a signature in its database, the system triggers an alert (detection) or blocks the traffic (prevention).

-Endpoint Protection Platforms (EPP)

Antivirus/anti-malware: EPP solutions use signature-based detection to identify and block known malware.

Function: They scan files and processes on endpoints, comparing them against a regularly updated database of malware signatures to prevent execution.

-Threat Intelligence Platforms (TIP)

Enrichment: TIPs collect and aggregate threat signatures and other intelligence from various sources.

Function: This information is then distributed to other security tools, such as firewalls and IDPS, to enhance their detection capabilities.

-Security Information and Event Management (SIEM)

Correlation: SIEMs don't use signatures for direct detection but ingest alerts and data from other security tools that do.

Function: They correlate security events from different sources, including signature-based alerts, to provide a more complete picture of a potential threat.

Attack Signatures and Negative Security

*The Negative Security Model

The Negative Security Model (also known as a Blacklist Model) is a fundamental approach to WAF security.

- Definition: It involves blocking all traffic that matches a predefined set of known malicious patterns or signatures. If the traffic does not match a bad pattern, it is allowed to pass.1
- WAF Role: The WAF's primary function in this model is to inspect traffic and discard anything on its "blacklist."
- Contrast (Positive Security): This model is the opposite of the Positive Security Model (Whitelist Model), which only allows traffic that matches a predefined set of known good behavior (e.g., only allowing numeric input in a phone number field). WAFs often use both models simultaneously for comprehensive protection.

Attack Signatures are the building blocks of the Negative Security Model.

- Definition: They are precise rules or patterns (often written as Regular Expressions or RegEx) designed to detect the telltale signs of common attack techniques within an HTTP request (URL, headers, parameters, or body).2
- Purpose: To identify threats like the command cmd.exe being injected into a parameter, or a specific string pattern used in a known remote code execution (RCE) exploit.
- Maintenance: Since hackers constantly develop new attacks, vendors (like F5) must continuously research and publish updates to the signature database.

Policy Attack Signature Sets: Categorization

The concept of Policy Attack Signature Sets refers to how the WAF organizes the thousands of available signatures to apply them efficiently and logically to different applications.3 Instead of applying every signature to every application, they are grouped:

Signature Set Category	Purpose and Target	Example Attack Types Included
SQL Injection	Targets databases (SQL, MySQL, Oracle) by blocking attempts to manipulate database queries.	UNION SELECT, OR 1=1, stacked queries.
Cross-Site Scripting (XSS)	Blocks attempts to inject client-side scripts (HTML/JavaScript) into web pages, often through input fields.	<script>, onerror=, alert(), <img> tags.
Command Execution	Blocks attempts to run unauthorized operating system commands on the web server.	cmd.exe, `
Log4Shell	A specific, critical set of signatures created to block attempts to exploit the Log4j vulnerability.	jndi:ldap://, specific serialized object injection patterns.
Web Access (or Outlook Web Access)	Signatures specifically tailored to protect common applications like Microsoft Exchange (Outlook Web Access), SharePoint, or specific content management systems (CMS).	Known vulnerabilities unique to those applications' structures.

Technical Advantage

By using categorized sets, WAFs can achieve two key goals:

1. Performance: The WAF only runs the relevant signature checks.4 For example, if an application doesn't use SQL, the administrator can disable the SQL Injection signature set, reducing processing overhead.5

2. Accuracy (Fewer False Positives): Application-specific sets (like those for Outlook) help ensure that the WAF doesn't block legitimate, but unusual, traffic that is common only to that particular application.

"More attack signatures used, we are going to more block requests"

That statement, describes the direct, and sometimes risky, relationship between the size of a WAF's signature database and its blocking actions.

Here is a detailed explanation of the implications of increasing the number of attack signatures:

-Impact of Increasing Attack Signatures

When a Web Application Firewall (WAF) uses more attack signatures, it means the firewall is configured to inspect traffic against a larger set of known malicious patterns.

1. Increased True Positives (Better Security)

Benefit: The primary goal is to increase the number of true positives. A true positive is a successful block of an actual attack.

Result: With more signatures, the WAF is better equipped to recognize a wider variety of threats, including obscure attacks, zero-day exploit variants, and techniques targeting less common platforms. This directly leads to more malicious requests being blocked, fulfilling the goal of enhanced security.

2. Increased False Positives (Business Risk)

Risk: The biggest risk of using too many signatures is the corresponding increase in false positives. A false positive occurs when the WAF mistakenly blocks a legitimate user request because the request's URL, parameter, or header happened to contain a string that matched an attack signature (RegEx).

Impact: False positives can disrupt business operations, as they can prevent users from completing purchases, submitting forms, or even logging in. This damages the user experience and can lead to lost revenue.

3. Increased Resource Consumption (Performance Cost)

Technical Cost: Every request that passes through the WAF must be scanned against the entire set of active attack signatures.

Performance Impact: A larger signature database means the WAF must dedicate more CPU and memory resources to complete the inspection before forwarding the request. This can introduce latency (delay) and reduce the overall request-per-second (RPS) throughput the WAF can handle. In high-traffic environments, this can become a significant performance bottleneck.

- Best Practice: Selective Tuning

Because of the trade-off between security (blocking more threats) and availability/performance (blocking fewer legitimate users), security teams rarely enable all available signatures.

Instead, they follow a process called Tuning:

Staging Mode: Initially, new signatures are often enabled in a learning or passive mode (e.g., "Alert" only, no "Block").

Analysis: The team monitors the logs to see which signatures are firing and determines if the triggered requests are legitimate or malicious.

-Refinement:

Disable/Modify: They disable signatures that generate excessive false positives for the specific application environment.

Enforce: They set highly effective, non-false-positive signatures to the "Block" action.

Conclusion: Using more attack signatures definitely leads to more blocked requests, but a successful WAF implementation is defined by blocking more bad requests while maintaining a near-zero rate of blocking good requests.

Parameters of attack signatures

Parameters are the specific, structured pieces of information that define, categorize, and instruct the WAF (Web Application Firewall) on how to detect and handle a specific threat.

These parameters are generally contained within the WAF's security policy database (e.g., the F5 BIG-IP ASM signature file).

Here are the key parameters that define an attack signature:

Key Parameters of Attack Signatures

Parameter (English)	Description
Signature ID	A unique numerical or alphanumeric identifier assigned to that specific attack pattern. (Essential for tracking)
Attack Type / Class	The broad security category the attack falls under.
	Examples: SQL Injection, Cross-Site Scripting (XSS), Command Execution, Buffer Overflow, Information Leakage.

Parameter (English)	Description
Pattern / Rule (RegEx)	The core of the signature. This is the Regular Expression that the WAF engine scans traffic against to find the malicious sequence of characters.
CVE Reference	If the signature relates to a public vulnerability, this field links it to the Common Vulnerabilities and Exposures ID.
	Example: CVE-2021-44228 (Log4Shell).
Context / Location	Specifies where in the HTTP request the WAF should look for the pattern.
	Examples: URL Path, Request Header, Parameter Value, Cookie Name, Raw Payload.
Risk / Severity	An assigned level indicating the potential damage if the attack is successful.
	Examples: High (Critical), Medium, Low.
Confidence Level	An indicator of how likely the signature is to be triggered by a legitimate request (False Positive). Higher confidence means less risk of blocking good traffic.
Action	The default response the WAF should take when the signature is matched.
	Examples: Block, Alert (Log Only), Ignore (Disabled).
System Version Scope	Specifies which versions of an application or operating system are vulnerable to this attack. Helps administrators target protection.
Update Timestamp	The date and time the signature was created or last modified by the vendor.

How These Parameters Work

When an administrator applies an attack signature policy, they are essentially telling the WAF:

"For Signature ID 1234, which is a High Severity SQL Injection targeting the Parameter Value (Context), use the RegEx pattern provided, and if you find a match, immediately Block the request (Action).

Creating and deploy a custom attack signature

"We are currently experiencing a brand-new attack (a zero-day or novel exploit). To mitigate this threat, I need to create and deploy a custom attack signature."

We must immediately develop a custom signature to block this specific traffic pattern."

*Note on Custom Attack Signatures: when an organization faces an attack that is not yet covered by vendor updates (a zero-day or a highly targeted variation), creating a custom attack signature is a critical defensive action.

Analysis: Security analysts examine the logs and packet data from the attack to identify the unique string or pattern (the payload).

Creation: They write a precise Regular Expression (RegEx) based on that pattern.

Deployment: This custom signature is then added to the WAF's security policy to immediately block any incoming traffic matching that specific malicious pattern.

A custom attack signature RegEx (Regular Expression) is the core technical component used in a WAF (Web Application Firewall) to identify and block a novel, unpatched, or highly specific attack against your application.

It must be crafted with extreme precision to match the malicious pattern while avoiding legitimate user input.

Here is a breakdown of the key elements, common syntax, and examples for creating an effective custom signature RegEx.

***Key RegEx Elements for WAF Signatures**

WAFs typically use a standard RegEx engine (like PCRE or RE2) and apply the pattern to the specific context (header, parameter, body).

RegEx Element	Description	Example	WAF Goal
\s	Matches any single whitespace character (space, tab, newline).	user\s*select	Allows zero or more spaces between keywords.
**`	`**	Logical OR operator. Matches one expression OR the other.	`(union
()	Groups characters together to apply quantifiers or operators.	(etc/passwd)	Treats this as a single unit to match.
[]	Character set. Matches any one character within the brackets.	[a-zA-Z]	Matches any single uppercase or lowercase letter.

RegEx Element	Description	Example	WAF Goal
?	Quantifier. Matches the preceding element zero or one time.	\s?	Allows zero or one space.
*	Quantifier. Matches the preceding element zero or more times.	\s*	Allows any amount of spacing (including none).
+	Quantifier. Matches the preceding element one or more times.	\d+	Requires at least one digit.
\	Escape character. Used to match a literal symbol (like a dot or parenthesis).	\.exe	Matches the literal string ".exe".

Custom RegEx Examples by Attack Type

The goal of these examples is to be specific enough to catch the attack but general enough to prevent easy evasion.

A. Blocking a Specific Zero-Day Exploit String

If you observe an attack attempting to send the payload eval(base64_decode('...')) in a specific parameter named input_data.

Parameter	Example RegEx	Explanation
Context	Parameter Value (for input_data)	Targets the malicious string only in a certain place.
Pattern	eval\s*\(\s*base64_decode\s*\(	Matches the command eval immediately followed by an opening parenthesis, allowing for any amount of spacing (\s*) or no spacing at all, followed by base64_decode and another opening parenthesis. This pattern is highly suspicious.

B. Blocking Obfuscated Command Execution (Linux)

To prevent an attacker from executing shell commands by chaining them with common delimiters.

Parameter	Example RegEx	Explanation
Context	Request URL or Parameters	Targets attempts to inject operating system commands.
Pattern	`(/etc/passwd	\s(bash

C. Blocking Obfuscated SQL Injection (Basic)

To catch common attempts to bypass the WAF using single quotes and comments.

Parameter	Example RegEx	Explanation
Context	Parameter Value	Focuses on user-submitted data likely destined for a database.
Pattern	`('	%27)\s*(--

* Best Practices for RegEx Signatures

1. Use Testing Tools: Always test your RegEx pattern thoroughly on a site like Regex101 or your WAF's built-in testing tool to confirm it matches the attack payload but *fails* to match legitimate data.
2. Be Precise (Avoid False Positives): Every character matters. Overly broad rules (like matching a single quote ' in isolation) will block huge amounts of legitimate traffic. Always anchor the malicious string to surrounding keywords or characters.
3. Target the Context: Don't apply a highly specific RegEx to the entire request if you know the exploit only appears in the User-Agent header or a specific parameter. Targeting the context reduces both False Positives and WAF processing load.
4. Use \s* Judiciously: Using \s* (zero or more spaces) is a good way to bypass attacker obfuscation, but it can also increase complexity and slow down the RegEx engine.

To configure a custom signature in FortiGate

1. Go to **Web Protection > Known Attacks > Custom Signature**. To access this part of the web UI, your administrator's account access profile must have **Read** and **Write** permission to items in the **Web Protection Configuration** category. For details, see Permissions.
2. From the **Custom Signature** tab, click **Create New**, then configure these settings:

Name	Type a unique name that can be referenced in other parts of the configuration. The maximum length is 63 characters.
Direction	Select which direction FortiWeb applies the expression to: • **Request**—The custom signature is designed to detect attacks. • **Response**—The custom signature is designed to detect information disclosure.
Action	Select the action FortiWeb takes when it detects a violation of the rule: • **Alert**—Accept the request and generate an alert email and/or log message. **Note:** If Direction is **Data Leakage**, does **not** cloak, except for removing sensitive headers. Sensitive information in the body remains unaltered. • **Alert & Deny**—Block the request (reset the connection) and generate an alert and/or log message. This option is applicable only if Direction is **Signature Creation**. You can customize the web page that FortiWeb returns to the client with the HTTP status code. For details, see Customizing error and authentication pages (replacement messages). • **Erase & Alert**—Hide replies with sensitive information (sometimes called "cloaking"). Block the reply (or reset the connection) or remove the sensitive information, and generate an alert email and/or log message. This option is applicable only if Direction is **Data Leakage**. If the sensitive information is a status code, you can customize the web page that will be returned to the client with the HTTP status code. **Note:** This option is not fully supported in Offline Protection mode. Effects will be identical to **Alert**; sensitive information will not be blocked or erased. • **Period Block**—Block subsequent requests from the client for a number of seconds. Also configure Block Period. You can customize the web page that FortiWeb returns to the client with the HTTP status code. For details, see Customizing error and authentication pages (replacement messages). **Note:** If FortiWeb is deployed behind a NAT load balancer, when using this option, you **must** also define an X-header that indicates the original client's IP. Failure to do so may cause FortiWeb to block **all** connections when it

	detects a violation of this type. For details, see Defining your proxies, clients, & X-headers. • **Erase, no Alert**—Hide replies with sensitive information (sometimes called "cloaking"). Block the reply (or reset the connection) or remove the sensitive information without generating an alert email and/or log message. This option is applicable only if Direction is **Data Leakage**. **Note:** This option is not fully supported in Offline Protection mode. • **Send HTTP Response**—Block and reply to the client with an HTTP error message and generate an alert email and/or log message. You can customize the attack block page and HTTP error code that FortiWeb returns to the client. For details, see Customizing error and authentication pages (replacement messages).
Block Period	Type the number of seconds that you want to block subsequent requests from the client after the FortiWeb appliance detects that the client has violated the rule. This setting is available only if Action is set to **Period Block**. The valid range is from 1 to 3,600 seconds (1 hour). For details, see Blocked IPs.
Severity	When rule violations are recorded in the attack log, each log message contains a **Severity Level** (severity_level) field. Select which severity level the FortiWeb appliance will use when it logs a violation of the rule: • Low • Medium • High The default value is **High**.
Trigger Action	Select which trigger, if any, that the FortiWeb appliance will use when it logs and/or sends an alert email about a violation of the rule. For details, see Blocked IPs.

Threat Weight	Set the weight for the threat by dragging the bar.

3. Click OK.
4. Click **Create New** to create a custom signature condition rule. The condition rules in the same custom signature are in "AND" relationship.
5. Complete the following settings:

Match Operator	• **Regular expression match**—The signature matches when the value of a selected target in the request or response matches the **Regular Expression** value. • **Greater than/Less than/Not equal/Equal**—FortiWeb determines whether the signature matches by comparing the value of a selected target in the request or response to the **Threshold** value.
Case Sensitive	Select to differentiate between upper case and lower case letters in the Regular Expression value. For example, when this option is enabled, an HTTP request involving tomcat would **not** match a sensitive information signature that specifies **T**omcat (difference is lower case "t").
Regular Expression	Specifies the value to match in a selected target. If the Action is **Alert & Erase**, enclose the portion of the regular expression to erase in brackets. For example, the regular expression value (webattack) detects and erases the string webattack from responses. To create and test a regular expression, click the >> (test) icon. For details, see Regular expression syntax.

Threshold	If Greater Than, Less Than, Equal, or Not Equal is selected as the Match Operator, this is the value that FortiWeb uses to evaluate a selected target.
Available Target/Selected Target	Use the arrows to add or remove locations in the HTTP request that FortiWeb scans for a signature match, then click the right arrow to move them into the **Search In** area. The argument's name and value are often included in the request body. In this case, you can't create a rule for the REQUEST_BODY target to detect the argument's name and value. Instead, you need to create rules for ARGS_NAME or/and ARGS_VALUE targets. For example, if you want to block the parameter count if its value is true ("count":true), you can create the following two rules: Rule #1: • Regular expression:count • Selected Target: ARGS_NAMES Rule #2: • Regular expression:true • Selected Target: ARGS_VALUE Whether a string should be treated as an argument or request body depending on the syntax of the content. For example, the above mentioned "count":true is only considered as argument in JSON and XML content types. For other content types, it is just a text string in the request body.

6. Click **OK**.
7. Repeat this procedure for each rule that you want to add. Click **Check Redundancy** to check redundant custom signature rules.
8. Click **OK** to save your custom signature.
9. Go to **Web Protection > Known Attacks > Custom Signature**. To access this part of the web UI, your administrator's account access profile must

have **Read** and **Write** permission to items in the **Web Protection Configuration** category. For details, see Permissions.

10. From the **Custom Signature Group** tab, click **Create New** to create a new group of custom signatures. Alternatively, to add your custom signature to an existing set, click Edit to add it to that set.
    The custom signatures in the same group are in "OR" relationship.
11. In **Name**, type a name that can be referenced by other parts of the configuration. The maximum length is 63 characters.
12. Click **OK**.
13. Click **Create New** to include individual rules in the set.
14. From the **Custom Signature** drop-down list, select a custom signature to add to the group. To view or change information associated with the custom signature, select the **Detail** link. The **Edit Custom Signature** dialog appears. You can view and edit the rules. Use the browser **Back** button to return.
15. Click **OK**.
16. Repeat the previous steps for each individual rule that you want to add to the custom signature set.
17. Group the custom signature set in a signature rule. For details, see Known Attacks .

When the custom signature set is enabled in a signature rule policy, you can add either the group or an individual custom signature rule in the group to an advanced protection custom rule. For details, see Custom Policy.

Example: ASP .Net version & other multiple server detail leaks

Example.com is a cloud hosting provider. Because it must offer whatever services its customers' web applications require, its servers run a variety of platforms—even old, unpatched versions with known vulnerabilities that have not been configured securely. Unfortunately, these platforms advertise their presence in a variety of ways, identifying weaknesses to potential attackers.

HTTP headers are one way that web server platforms are easily fingerprinted. Example.com wants to remove unnecessary headers that provide server details to clients in order to make it harder for attackers to fingerprint their platforms and craft successful attacks. Specifically, it wants to erase these HTTP response headers:

X-AspNet-Version: 2.0.50727

X-AspNetMvc-Version: 3.0

Server: Microsoft-IIS/7.0

X-Powered-By: ASP.NET

To do this, Example.com writes a custom signature that erases content with 4 meet condition rules, one to match the contents of each header (but not the header's key), and includes the custom signature in the signature set used by the protection profile:

Direction	Response
Action	Alert & Erase
Severity	Low
Trigger Action	notification-servers1
Meet condition rule 1	
Match Operator	Regular expression match
Regular Expression	\bServer:(.*)\b
Selected Target	ARGS_NAMES
Meet condition rule 2	
Match Operator	Regular expression match
Regular Expression	\bX-AspNetMvc-Version:(.*)\b
Selected Target	ARGS_NAMES
Meet condition rule 3	
Match Operator	Regular expression match
Regular Expression	\bX-AspNet-Version:(.*)\b
Selected Target	ARGS_NAMES

Meet condition rule 4	
Match Operator	Regular expression match
Regular Expression	\bX-Powered-By:(.*)\b
Selected Target	ARGS_NAMES

The result is that the client receives HTTP responses with headers such as:

Server: XXXXXXXX

X-Powered-By: XXXXXXXX

X-AspNet-Version: XXXXXXXX

The first new XSS attack found was:

<img

src='/images/nonexistant-file'

onerror= document.write(

<scr I pt src= www.example.co/xss.js>);

/>

The above attack works by leveraging a client web browser's error handling against itself. Without actually naming JavaScript, the attack uses the JavaScript error handling event onError() to execute arbitrary code with the HTML <img> tag. The <img> tag's source is a non-existent image. This triggers the web browser to load an arbitrary script from the attacker's command-and-control server. To avoid detection, he attacker has even bought a DNS name that looks like one of example.com's legitimate servers: www.example.co.

The incident response team has also found two other classes of XSS that evades the forum's own XSS sanitizers (which only look for injection of <script> and <object> tags). The first one exploits a web browser's parser by tricking it with additional quotes in an unexpected place:

<img """><script>alert("XSS")</script>">

The second one exploits the nature of all web pages with images and other external files. Other than the web page itself, all images, scripts, styles, media, and objects cause the web browser to make secondary HTTP requests: one for each component of the web page. Here, the <img> tag causes the client's web browser to make a request that is actually an injection attempt on another website.

<img src="http://other.example.com/command.php?variable=attackcode">

The incident response team has written 3 regular expressions to detect each of the above XSS attack classes, as well as similar permutations that use HTML tags other than <img>:

<(.*)src(\s)*=(\s)*[‘’“”](\s)*(.*)(\s)*[‘’“”](\s)*onError

<(.*)[‘’“”][‘’“”]*(.*)>(\s)*<script>

<(\s)*[^(<script)](\s)*src(\s)*=(\s)*(HTTP|HTTPS|ftp|\\\\\|\/\/)(.*)\?

To check for any of the 3 new attacks, the team creates a custom signature with 3 meet condition rules. (Alternatively, the team can create a single meet condition rule that joins the 3 regular expressions by using pipe (|) characters between them.)

Conclusion:

Attack signatures are the digital fingerprints of known cyber threats, serving as the sentinel against malicious activities targeting web applications. The WAF utilizes a signature pool comprising both vendor-supplied (System-Supplied) and custom User-Defined Signatures to perform deep inspection and block requests that match a predefined malicious pattern.

The effectiveness of this security mechanism lies in its structured three-step technical process: Creation, Distribution, and Inspection. The vendor continuously converts new vulnerability patterns into precise Regular Expressions (RegEx) , packages them into updates (e.g., the *.im file) , and the WAF enforces them by terminating the request upon a match.

While increasing the number of signatures enhances security by boosting True Positives (blocking actual attacks) , it carries significant risks of generating False Positives (blocking legitimate users) and increasing Resource Consumption (latency). Therefore, managing attack signatures requires selective tuning and adherence to best practices to maintain a crucial balance between security and application availability.

Recommendations:

To maximize the efficacy and security posture provided by Attack Signatures, security administrators should adhere to the following recommendations:

Prioritize Policy Tuning over Bulk Activation

Do not enable all available signatures. Use Policy Attack Signature Sets (e.g., SQL Injection, XSS) to logically group and apply only the protection relevant to the specific application.

Utilize Staging Mode or "Alert" mode first for new signatures to monitor logs and identify false positives before setting the action to Block.

Ensure Consistent Signature Updates

Set up automated or mandatory processes to download and install vendor updates (e.g., F5's *.im file) as soon as they are released to provide immediate protection against new CVEs and zero-day threats.

Master Custom Signature Creation

When facing a zero-day or proprietary attack, immediately develop a custom signature.

Ensure the custom RegEx pattern is precise and highly specific to the attack payload , targeting only the Context/Location (e.g., a specific parameter) to reduce the risk of false positives.

Always test the RegEx pattern with both malicious and legitimate traffic before deployment.

Targeted Context Enforcement

When configuring a signature, utilize the Context/Location parameter to specify where the WAF should look (e.g., Parameter Value, Request Header). This is vital for reducing the WAF processing load and improving both performance and accuracy.

Integrate Signature Alerts into SIEM

Ensure that "Violation attack signature detected" alerts are ingested by the SIEM (Security Information and Event Management) platform to correlate security events and provide a complete picture of ongoing threats.

References:

1- MyF5 Home, Knowledge Centers, BIG-IP ASM BIG-IP Application Security Manager: Implementations, Working with Attack Signatures.

2- Discovering Attack Signature and Its Travel Path using

Graphical Model in CPS: A Case Study

PRANEETA MAGANTI, PARESH SAXENA, and RAJIB RANJAN MAITI, Department of

Computer Science and Information Systems, Birla Institute of Technology and Science Pilani,

Hyderabad Campus, Hyderabad, India

3- YouTube, F5 3-day BIG-IP ASM / AWAF Course - Lesson 5 - Attack Signatures

4- FORTINET, Document Library, Defining custom data leak & attack signatures.