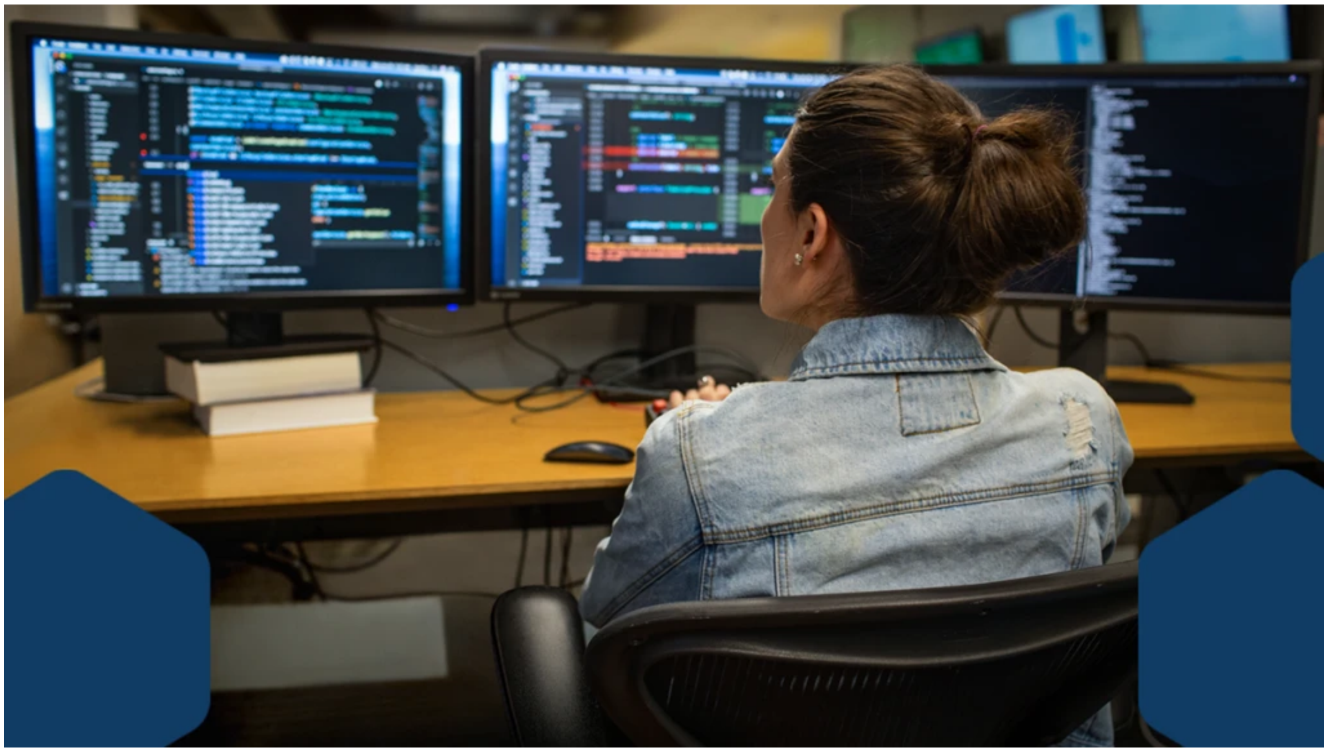
Code injection attacks using publicly disclosed ASP.NET machine keys

microsoft.com/en-us/security/blog/2025/02/06/code-injection-attacks-using-publicly-disclosed-asp-net-machine-keys/

February 6, 2025



By

In December 2024, Microsoft Threat Intelligence observed limited activity by an unattributed threat actor using a publicly available, static ASP.NET machine key to inject malicious code and deliver the Godzilla post-exploitation framework. In the course of investigating, remediating, and building protections against this activity, we observed an insecure practice whereby developers have incorporated various publicly disclosed ASP.NET machine keys from publicly accessible resources, such as code documentation and repositories, which threat actors have used to perform malicious actions on target servers.

Microsoft has since identified over 3,000 publicly disclosed keys that could be used for these types of attacks, which are called ViewState code injection attacks. Whereas many previously known ViewState code injection attacks used compromised or stolen keys that are often sold on dark web forums, these publicly disclosed keys could pose a higher risk because they are available in multiple code repositories and could have been pushed into development code without modification.

Microsoft recommends that organizations do not copy keys from publicly available sources and to regularly rotate keys. Microsoft Defender for Endpoint can help reduce this risk by detecting publicly disclosed keys. To further discourage this practice, we have also removed key samples from limited instances where they were included in our own public documentation.

The limited malicious activity observed in December 2024 identified one publicly disclosed key that was used to inject malicious code. Microsoft Threat Intelligence continues to monitor the additional use of this attack technique. In this blog, we share more information about ViewState code injection attacks and provide recommendations for securing machine keys and monitoring configuration files.

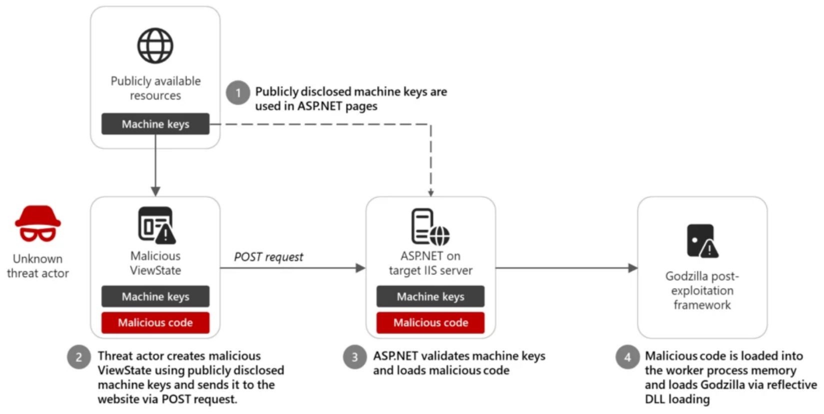
What are ViewState code injection attacks?

ViewState is the method by which ASP.NET Web Forms preserve page and control state between postbacks. ViewState data is stored in a hidden field on the page and is encoded using Base64-encoding. To protect ViewState against tampering and information disclosure, the ASP.NET page framework uses machine keys: ValidationKey and DecryptionKey. ValidationKey is used to create a message authentication code (MAC) to be attached in the ViewState. DecryptionKey is related to the option of encrypting ViewState. These keys are either auto-generated and stored in registry or specified manually in config files.

If these keys are stolen or made accessible to threat actors, these threat actors can craft a malicious ViewState using the stolen keys and send it to the website via a POST request. When the request is processed by ASP.NET Runtime on the targeted server, the ViewState is decrypted and validated successfully because the right keys are used. The malicious code is then loaded into the worker process memory and executed, providing the threat actor remote code execution capabilities on the target IIS web server.

ViewState code injection attack leading to Godzilla post-exploit framework

In December 2024, an unattributed threat actor conducted a ViewState code injection attack leveraging a publicly known machine key. The malicious ViewState payload reflectively loaded *assembly.dll* (SHA-256: 19d87910d1a7ad9632161fd9dd6a54c8a059a64fc5f5a41cf5055cd37ec0499d), a Godzilla post-exploitation framework, followed by plugin modules. Godzilla's functionality includes executing malicious commands, injecting shellcode into processes, and more.


Publicly available resources
Machine keys
1 Publicly disclosed machine keys are used in ASP.NET pages
Unknown threat actor
Malicious ViewState
Machine keys
Malicious code
POST request
ASP.NET on target IIS server
Machine keys
Malicious code
Godzilla post-exploitation framework
2 Threat actor creates malicious ViewState using publicly disclosed machine keys and sends it to the website via POST request.
3 ASP.NET validates machine keys and loads malicious code
4 Malicious code is loaded into the worker process memory and loads Godzilla via reflective DLL loading

ViewState code injection attack chain leading to Godzilla.

Recommendations

Microsoft Defender for Endpoint customers can identify publicly disclosed keys in their environment based on the presence of the alert *Publicly disclosed ASP.NET machine key*. This alert is for informational purposes and is not indicative of attack activity. Additionally, Microsoft has provided a list of hash values for identified publicly disclosed machine keys in our Github repository and recommends checking machine keys in your environment using the provided script.

If publicly disclosed keys are identified in your environment, Microsoft recommends the following actions, depending on your scenario:

- If you are running an ASP.NET web application on .NET Framework and this is not part of Exchange Server or SharePoint, review the following potential setups:
    - You have set fixed key values using the *<machineKey>* element of the *web.config* to align the encryption/decryption of ViewState values within multiple servers that are part of a web farm. This configuration assumes you have multiple web servers hosting the same instance of a web application to distribute load among them, and that requests that were originally served from one server in the farm could trigger POST requests to another server of the same farm.
        - In this case, you will need to rotate the values of your machine keys in all servers of the farm, either by using the IIS manager console or PowerShell (see details below). Ensure that you use the same newly generated values on all servers in the farm.
    - You have set fixed key values using the *<machineKey>* element for a single server that is running your ASP.NET web application. In this scenario, removing the *<machineKey>* element from the configuration will revert the application to the auto-generated values for the ASP.NET machine keys that are stored inside your computer's registry.
        - See the below details on how the *<machineKey>* element can be removed by editing the *web.config* configuration file or using the IIS manager console.
- If you are using SharePoint or Exchange web applications, SharePoint uses its own key management system, which allows the keys to be rotated as described in this article.

Follow the steps below for removing or replacing the ASP.NET machine key values in the *web.config* configuration file using either the IIS manager console or PowerShell.

Using the IIS manager console:

1. From the IIS manager console, select the website or web application that contains the fixed key values in the *web.config* configuration file.
2. From the middle pane of the IIS manager console, select the **Machine Key** element icon.
3. To create new machine key values, select the **Generate Keys** button on the right-hand side of the console, which will populate new values for *Validation* and *Decryption* key textboxes.
4. Once the new values are populated, select the **Apply** button from the right-hand side pane to persist the new values into the *web.config* file for the target website or web application.
5. To remove the fixed keys and rely on the auto-generated machine key values for ASP.NET, select the **Automatically generate at runtime** checkboxes for both the *Validation* and *Decryption* keys. This will render the two text fields for these values disabled.
6. Proceed to select the **Apply** button on the right-hand side of the console, which will remove the *<machineKey>* element from the *web.config* file of the website or web application.

Using PowerShell:

Using PowerShell, create a .ps1 file (for example, *GenerateKeys.ps1*) with the following content:

```
function Generate-MachineKey {
  [CmdletBinding()]
  param (
    [ValidateSet("AES", "DES", "3DES")]
    [string]$decryptionAlgorithm = 'AES',
    [ValidateSet("MD5", "SHA1", "HMACSHA256", "HMACSHA384", "HMACSHA512")]
    [string]$validationAlgorithm = 'HMACSHA256'
  )
  process {
    function BinaryToHex {
        [CmdLetBinding()]
        param($bytes)
        process {
            $builder = new-object System.Text.StringBuilder
            foreach ($b in $bytes) {
              $builder = $builder.AppendFormat([System.Globalization.CultureInfo]::InvariantCulture, "
{0:X2}", $b)
            }
            $builder
        }
    }
    switch ($decryptionAlgorithm) {
      "AES" { $decryptionObject = new-object System.Security.Cryptography.AesCryptoServiceProvider }
      "DES" { $decryptionObject = new-object System.Security.Cryptography.DESCryptoServiceProvider }
      "3DES" { $decryptionObject = new-object System.Security.Cryptography.TripleDESCryptoServiceProvider
}
    }
    $decryptionObject.GenerateKey()
    $decryptionKey = BinaryToHex($decryptionObject.Key)
    $decryptionObject.Dispose()
    switch ($validationAlgorithm) {
      "MD5" { $validationObject = new-object System.Security.Cryptography.HMACMD5 }
      "SHA1" { $validationObject = new-object System.Security.Cryptography.HMACSHA1 }
      "HMACSHA256" { $validationObject = new-object System.Security.Cryptography.HMACSHA256 }
      "HMACSHA385" { $validationObject = new-object System.Security.Cryptography.HMACSHA384 }
      "HMACSHA512" { $validationObject = new-object System.Security.Cryptography.HMACSHA512 }
    }
    $validationKey = BinaryToHex($validationObject.Key)
    $validationObject.Dispose()
    [string]::Format([System.Globalization.CultureInfo]::InvariantCulture,
      "&lt;machineKey decryption=`"{0}`" decryptionKey=`"{1}`" validation=`"{2}`" validationKey=`"{3}`"
/>",
      $decryptionAlgorithm.ToUpperInvariant(), $decryptionKey,
      $validationAlgorithm.ToUpperInvariant(), $validationKey)
  }
}
```

Using a Windows PowerShell command prompt, navigate to the location containing your .ps1 file. Load the .ps1 file (for example, *GenerateKeys.ps1*) by executing the following command:

```
..\GenerateKeys.ps1
```

Call the function within the script file by executing the following command:

```
Generate-MachineKey
```

Copy the resulting *<machineKey>* element to the *web.config* file of your website or web application, replacing the existing one.

NOTE: If successful exploitation of publicly disclosed keys has occurred, rotating machine keys will not sufficiently address possible backdoors or persistence methods established by a threat actor or other post-exploitation activity, and additional investigation may be warranted. In particular, web-facing servers should be fully investigated and strongly considered for re-formatting and re-installation in an offline medium in cases where publicly disclosed keys have been identified, as these servers are most at risk of possible exploitation.

Microsoft also recommends the following best practices for securing machine keys and web servers:

- Follow secure DevOps standards and securely generate machine keys. Avoid using default keys or keys listed in public resources.
- At deployment, encrypt sensitive information like the *machineKey* and *connectionStrings* elements in *web.config*. This prevents these secrets from ever existing in plaintext on the file system, inhibiting an attacker's ability to read these secrets at all.
- Upgrade your application to use ASP.NET 4.8 to enable Antimalware Scan Interface (AMSI) capabilities.
- Harden Windows Servers instances by using attack surface reduction rules such as Block Webshell creation for Servers. Attack surface reduction rules are sweeping settings that are effective at stopping entire classes of threats.

Microsoft Defender XDR detections

Microsoft Defender XDR customers can refer to the list of applicable detections below. Microsoft Defender XDR coordinates detection, prevention, investigation, and response across endpoints, identities, email, apps to provide integrated protection against attacks like the threat discussed in this blog.

Customers with provisioned access can also use Microsoft Security Copilot in Microsoft Defender to investigate and respond to incidents, hunt for threats, and protect their organization with relevant threat intelligence.

Microsoft Defender Antivirus

Microsoft Defender Antivirus detects post-exploitation activity such as delivery of the Godzilla framework with the following components as the following malware. Note, however, that these alerts can also be triggered by unrelated threat activity and are not necessarily indicative of ViewState code injection attacks:

- Backdoor:MSIL/GodZillaMod.A
- Trojan:Win32/WebShellTerminal
- Backdoor:MSIL/Godzela

Microsoft Defender for Endpoint

The following Microsoft Defender for Endpoint alert indicates the presence of publicly disclosed machine keys but are not indicative of exploitation activity. To get this alert, you must be running Microsoft Defender Antivirus as your active antivirus. Customers who receive this alert should review our steps for rotating or removing machine keys in the Recommendations section.

> Publicly disclosed ASP.NET machine key

The following alert might also indicate threat activity such as code injection attacks. Note, however, that this alert can be also triggered by unrelated threat activity.

> IIS worker process loaded suspicious .NET assembly

Microsoft Security Copilot

Security Copilot customers can use the standalone experience to create their own prompts or run the following pre-built promptbooks to automate incident response or investigation tasks related to this threat:

- Incident investigation
- Microsoft User analysis

- Threat actor profile
- Threat Intelligence 360 report based on MDTI article
- Vulnerability impact assessment

Note that some promptbooks require access to plugins for Microsoft products such as Microsoft Defender XDR or Microsoft Sentinel.

Hunting queries

Microsoft recommends monitoring configuration files and configurable locations using the following audit policy settings, which will enable the logging of Event ID 4663 in the Windows Security Event Log. Before setting up auditing, note that:

- There are different scopes to configuration settings as some have a global scope and some are effective only to the scope of the application, the root *web.config* file, or the *machine.config* file. Each scope has the potential to have the <*machineKey*> section declared, possibly containing secrets.
- The scope of a configuration section is defined in the *allowDefinition* attribute of the section Element for *configSections* (General Settings Schema) element in the *machine.config* file for all sections that are included with ASP.NET. The scope of a configuration setting is listed for each element in ASP.NET Configuration Settings and General Configuration Settings (ASP.NET), in the **Element Information** table next to **Configurable Locations**.
- In configurations where it is specified to auto-generate keys at runtime, keys are stored in the application pool identity registry hive.
- It is important to understand that these locations also may be called through the Volume Shadow Copy Service.

Monitor access to ASP.NET configuration files by configuring Advanced Audit Policy settings:

1. Open the Group Policy Management Console from **Server Manager** > **Tools** > **Group Policy Management**.
2. Open a GPO that will apply policy to the ASP.NET application or web server.
3. From the window that opens, go to **Computer Configuration** > **Policies** > **Windows Settings** > **Security Settings** > **Local Policies** > **Audit Policy** > **Audit object level access**.
   - Define these policy settings: **Checked**
   - Success: **Checked**
   - Failure: Optional

After enabling Audit object access, enable auditing on individual files through the Security Tab on file properties to audit sensitive configuration files:

1. Locate your *web.config* or other configuration file of concern and open the **Security Tab** and select the **Advanced** button.
2. Select the **Auditing** tab and **Add** a new rule.
3. **Select a principal** and resolve the principal **Everyone**. Select the **Read** checkbox at minimum.
4. **Save** your changes by clicking **Ok, Ok, Ok.**

After **Audit object access** is configured, and **Auditing** has been turned on for specific files, alerts can be viewed using Event ID 4663 in the Windows Security Event Log. Information includes ***SubjectUserName*** accessing the file, and the ***ProcessId*** and ***ProcessName***. Seeing a username or process that is not the IIS Application Pool service account or the *w3wp.exe* process could be a sign of tampering and attempt to compromise a machine key.

Microsoft Sentinel

Microsoft Sentinel customers can use the TI Mapping analytics (a series of analytics all prefixed with 'TI map') to automatically match the malicious indicators mentioned in this blog post with data in their workspace. If the TI Map analytics are not currently deployed, customers can install the Threat Intelligence solution from the Microsoft Sentinel Content Hub to have the analytics rule deployed in their Sentinel workspace.

Additionally, once auditing is enabled for specific configuration files that may contain machine keys, Sentinel customers can leverage the *SecurityEvent* table to analyze collected Event ID 4663 and identify potential anomalies and unauthorized or suspicious file access attempts. To begin the threat hunting process, the following query can be used as a starting point:

```
SecurityEvent
| where TimeGenerated > ago(1d)
| where EventID == 4663
| where ObjectName contains "web.config"  or ObjectName contains "machine.config"
| summarize StartTime = max(Time Generated), EndTime = min(TimeGenerated), count() by EventID, Account,
Computer, Process, SubjectUserName, SubjectDomainName, ObjectName, ObjectType, ProcessName, ProcessId,
AccountType, AccessMask
```

Some of the key information captured in the result includes the *SubjectUserName*, which identifies the user accessing the file, along with the associated *ProcessId* and *ProcessName*. Observing a username or process other than the expected IIS Application Pool service account or the standard *w3wp.exe* process may signal potential tampering or an attempt to compromise the machine key.

Indicators of compromise

Indicator	Type	Description	First seen	Last seen
19d87910d1a7ad9632161fd9dd6a54c8a059a64fc5f5a41cf5055cd37ec0499d	SHA-256	Godzilla post-exploitation framework	2024-12-11	2024-12-19

Microsoft has provided a list of hash values for identified publicly disclosed machine keys in our Github repository. We recommend using the included script to compare the values against static keys in your environment to determine whether your machine keys have been disclosed in publicly accessible resources, and using the Recommendations listed above to rotate or remove machine keys.

References

https://www.zerodayinitiative.com/blog/2020/2/24/cve-2020-0688-remote-code-execution-on-microsoft-exchange-server-through-fixed-cryptographic-keys

Learn more

For the latest security research from the Microsoft Threat Intelligence community, check out the Microsoft Threat Intelligence Blog: https://aka.ms/threatintelblog.

To get notified about new publications and to join discussions on social media, follow us on LinkedIn at https://www.linkedin.com/showcase/microsoft-threat-intelligence, and on X (formerly Twitter) at https://x.com/MsftSecIntel.

To hear stories and insights from the Microsoft Threat Intelligence community about the ever-evolving threat landscape, listen to the Microsoft Threat Intelligence podcast: https://thecyberwire.com/podcasts/microsoft-threat-intelligence.

Get started with Microsoft Security

Microsoft is a leader in cybersecurity, and we embrace our responsibility to make the world a safer place.

Learn more