WEB-Application
FireWall

Release Version 2021.10.19.x for .Net 5.0, .Net 6.0, .Net Standard 2.1 and .Net Core 3.1

The getting started manual

TABLE OF CONTENT

Understanding the Walter.Web.FireWall framework **5**

How to Set-up the firewall in your application **6**

Source the binaries 6

Register your web application and get a license key 7

The registration process 7

Minimum configuration 8

Securing the web application using the firewall 9

Always protect using Always-On 9

Protecting specific endpoints 10

Securing your endpoints **11**

The BlockDuration Attribute 11

The GeoBlock Attribute 12

The GeoIgnore Attribute 14

The Ignore Attribute 14

The HitRatio Attribute 15

The Users Attribute 16

The WhiteListIP Attribute 20

The WhiteListSubnet Attribute 21

The ActionProtector Attribute 21

The RedirectRule Attribute 22

The CrossSite Attribute 23

The Disabled Attribute 24

Using browser-based protection 25

Inspecting and manual altering the endpoint protection 26

Reporting on the health of the firewall and view the activity 26

What to do when the firewall is blocking and you are not sure why 28

On remote server 28

You can use the Plugin Walter.Web.FireWall.EventLog to have the firewall log all incidents in an easily acceptable location without compromising security. 28

During development 28

Getting Blocked by your own BlockedPatterns 30

Taking control over the firewall 32

Your own IFireWall implementation by inheriting from the FireWallBase class 32

Events on the IFireWall interface and FireWallBase class 33

The OnResourceRequested event 33

The OnResourceSend event 34

The OnRootPageRequestDisposed event 34

Use events for advanced runtime configuration 37

The IFireWallConfig.OnRulesCreated event 37

The OnEndpointsCreated event 37

The OnFireWallCreated event 38

Events allowing you to control the firewall’s recommendations 39

The FireWallBase.OnIncident event 39

The FireWallBase.OnGuardAction event 40

The FireWallBase.OnUserTypeChange event 41

Reset FireWall incidents 42

The IFireWall.ResetFireWallBlocking method 42

The IFireWall.Clear method 43

IFireWall.ChangeStateTo method 43

Firewall stability 44

Access to Runtime Errors 44

Access to configuration errors 44

Using Firewall Plugins’ **45**

Geography 45

Walter.Web.FireWall.Geo.MaxMind 45

Walter.Web.FireWall.Geo.Native 46

Reporting 47

Walter.Web.FireWall.EventLog 47

Walter.Web.FireWall.SMTPLogger 48

Walter.Web.FireWall.SqlLogger 49

Walter.Web.FireWall.DiskLogger 50

Walter.Web.FireWall.ILogger 51

Data Sources 53

Walter.Web.FireWall.CookieStore 53

User-Agent guard module data 53

Native User-Agent recognition 53

Walter.Web.FireWall.UserAgent.UsersStack 54

Advanced protection on dedicated servers **55**

External Data Providers 57

Firewall data source NugetPackage Walter.Net.Windows.FireWall 57

The Walter.Net.Networking NuGet Package 57

The Walter.Net.FireWall.LookWhoisTalking NuGet Package 57

Documentation & Help **59**

Online-Help 59

Offline-Help 59

Integrating with Visual studio 60

Continuing from here... **61**

Understanding the Walter.Web.FireWall framework

The framework uses default rules as well as annotations and helpers to override or extend the default rules to control access to resources in a .Net application. At the resource level, like actions, pages or REST services. The firewall can be on by default or it can be used to protect only certain endpoints only. One uses this in endpoint protection the same way you would use the authorization attribute.

Inside a razor page, you can use attributes inside your normal HTML tags to condition if a tag is rendered for a given country or type of user. We try to classify users in Search Engines, Humans, or Automated Bots as soon as possible allowing you to adjust your content for it.

You can configure the rules in the FireWall Configuration as well as in annotation directly on your controllers or the actions in your controllers. Once a “violation” of the “rules” is detected you have the opportunity to:

A. Never reply to the request and make it time-out on the requestors side, this is effective in dealing with, and the blocking of bots. Bots that are scrubbing data normally deal with Http error codes quite well, having them wait slows them down dramatically. Not getting an error code does make it harder for the bots to work with your content,
B. Block the request and generate a 50x HTTP response, this saves CPU time and resources by just not answering malicious consumers. Also, bots may think the site is down.
C. Return a 40x HTTP response informing the consumer something went wrong.
D. Redirect the request to another location, this can be used to validate the user to content he is allowed to see or allows the user to interact with the system in a way you find appropriate.
E. Allow a user to violate a given rule for a fixed amount of times during a fixed period. This gives a request a “free pass” allowing you to make the firewall less strict and work with Framework HTML annotation to alter/ hide some of the content.

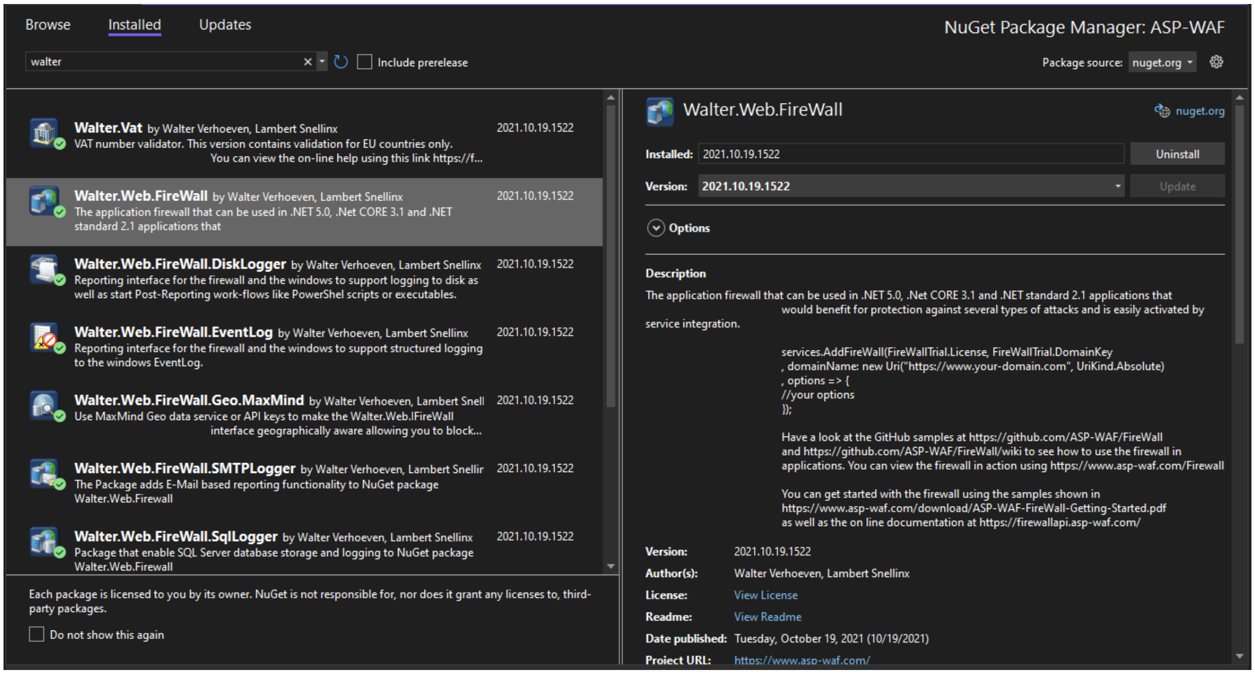
This document will show with several code samples on how to integrate the firewall framework and how easy this can be implemented using simple scenarios. We also cover the topic of reporting and troubleshooting when you are confronted with behaviours that you are not expecting.

How to Set-up the firewall in your application

The firewall framework consists of 2 main parts. Part 1 analyses the requesting source and monitors its activity passively. Part 2 does the same but does it actively by using javascript and can therefore only be used by web applications that have a user interface. This part of the document tells you how to set this up assuming you have a web application that has both REST API as well as server Pages to users where the users might be using either or both endpoints.

Source the binaries

Download the “Walter.Web.Firewall" NuGet Package. In this example, we will assume that we use Walter.Web.Firewall for .Net core web applications. This package supports .Net 6, .Net 5, .Net Core 3.1 as well as .Net standard 2.1. You can do this from any NuGet repository by entering Walter.Web.Firewall in the search box.



Walter.Vat by Walter Verhoeven, Lambert Snellinx 2021.10.19.1522
VAT number validator. This version contains validation for EU countries only.
You can view the on-line help using this link https://f...

Walter.Web.FireWall by Walter Verhoeven, Lambert Snellinx 2021.10.19.1522
The application firewall that can be used in .NET 5.0, .Net CORE 3.1 and .NET standard 2.1 applications that

Walter.Web.FireWall.DiskLogger by Walter Verhoeven, Lambert Snellinx 2021.10.19.1522
Reporting interface for the firewall and the windows to support logging to disk as well as start Post-Reporting work-flows like PowerShel scripts or executables.

Walter.Web.FireWall.EventLog by Walter Verhoeven, Lambert Snellinx 2021.10.19.1522
Reporting interface for the firewall and the windows to support structured logging to the windows EventLog.

Walter.Web.FireWall.Geo.MaxMind by Walter Verhoeven, Lambert Snell 2021.10.19.1522
Use MaxMind Geo data service or API keys to make the Walter.Web.IFireWall
interface geographically aware allowing you to block...

Walter.Web.FireWall.SMTPLogger by Walter Verhoeven, Lambert Snellir 2021.10.19.1522
The Package adds E-Mail based reporting functionality to NuGet package Walter.Web.Firewall

Walter.Web.FireWall.SqlLogger by Walter Verhoeven, Lambert Snellinx 2021.10.19.1522
Package that enable SQL Server database storage and logging to NuGet package Walter.Web.Firewall

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Walter.Web.FireWall

Installed: 2021.10.19.1522

Version: 2021.10.19.1522

Description

The application firewall that can be used in .NET 5.0, .Net CORE 3.1 and .NET standard 2.1 applications that would benefit for protection against several types of attacks and is easily activated by service integration.

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
, domainName: new Uri("https://www.your-domain.com", UriKind.Absolute)
, options => {
//your options
});
```

Have a look at the GitHub samples at https://github.com/ASP-WAF/FireWall and https://github.com/ASP-WAF/FireWall/wiki to see how to use the firewall in applications. You can view the firewall in action using https://www.asp-waf.com/Firewall

You can get started with the firewall using the samples shown in https://www.asp-waf.com/download/ASP-WAF-FireWall-Getting-Started.pdf as well as the on line documentation at https://firewallapi.asp-waf.com/

Version: 2021.10.19.1522
Author(s): Walter Verhoeven, Lambert Snellinx
License: View License
Readme: View Readme
Date published: Tuesday, October 19, 2021 (10/19/2021)
Project URL: https://www.asp-waf.com/

You can get started using the trial keys as demonstrated here. You are ready to configure the firewall after having integrated the NuGet binaries.

Register your web application and get a license key

To be able to use the firewall you need to register your domain and get a license. The registration wizard requires you to enter the company details as well as the full URL of the domain name. You need a license per domain as well as any subdomains that you may use.

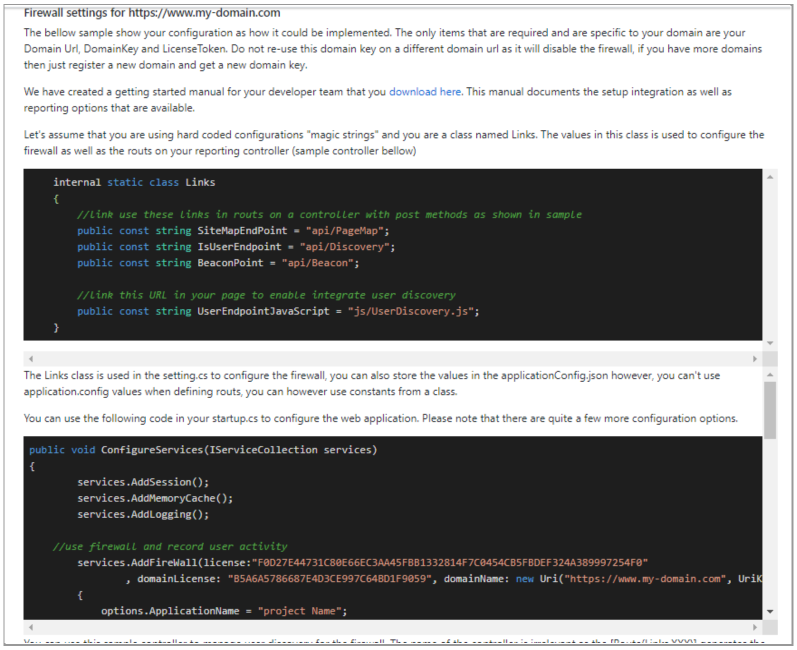
> We offer the binaries for firewall bundles in this download. Sometimes one does not have access to NuGet.org as your corporation may be blocking it, Or, you simply like to use local packages to keep change testing simple especially when having 1 version of the firewall in all corporate websites.

The registration process

When registering your domain the registration wizard will provide you with sample code that you can use to configure your web application.

If registering a community version you will only need to enter your domain URL. when licensing a paid version you will need to provide your company details. Licensed versions contain more features and allow the use of Add-Ons.

The below image shows a part of the registration wizard after having completed the community license registration. It might be a little small for this page, however, it will enter the license token and domain key in the sample code for the URL you specified during the registration.



Firewall settings for https://www.my-domain.com

The bellow sample show your configuration as how it could be implemented. The only items that are required and are specific to your domain are your Domain Url, DomainKey and LicenseToken. Do not re-use this domain key on a different domain url as it will disable the firewall, if you have more domains then just register a new domain and get a new domain key.

We have created a getting started manual for your developer team that you download here. This manual documents the setup integration as well as reporting options that are available.

Let's assume that you are using hard coded configurations "magic strings" and you are a class named Links. The values in this class is used to configure the firewall as well as the routs on your reporting controller (sample controller bellow)

```
    internal static class Links
    {
        //link use these links in routs on a controller with post methods as shown in sample
        public const string SiteMapEndPoint = "api/PageMap";
        public const string IsUserEndpoint = "api/Discovery";
        public const string BeaconPoint = "api/Beacon";

        //link this URL in your page to enable integrate user discovery
        public const string UserEndpointJavaScript = "js/UserDiscovery.js";
    }
```

The Links class is used in the setting.cs to configure the firewall, you can also store the values in the applicationConfig.json however, you can't use application.config values when defining routs, you can however use constants from a class.

You can use the following code in your startup.cs to configure the web application. Please note that there are quite a few more configuration options.

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddSession();
        services.AddMemoryCache();
        services.AddLogging();

    //use firewall and record user activity
        services.AddFireWall(license:"F0D27E44731C80E66EC3AA45FBB1332814F7C0454CB5FBDEF324A389997254F0"
                , domainLicense: "B5A6A5786687E4D3CE997C64BD1F9059", domainName: new Uri("https://www.my-domain.com", UriK
        {
            options.ApplicationName = "project Name";
```

You can use this sample controller to manage user discovery for the firewall. The name of the controller is irrelevant as the [Route(Links.XXX)] generates the

When licensing a paid license option you will be presented with the configuration of the Add-Ons' that are available in your license level.

Minimum configuration

At a bare minimum, you need to use the following for use with a web application's ConfigureServices in your Startup.cs file.

```
services.AddFireWall("Your Token", "Domain Key"
                , new Uri("https://www.MyDomain.com", UriKind.Absolute),
   options =>{
      //name used in reporting like email notifications
      options.ApplicationName = "www.MyDomain.com";

      options.Cypher.ApplicationPassword = "Pa$$w©rd";
      // configure the firewall for pages as well as api requests
      options.FireWallMode = FireWallModes.WebSiteWithApi;
   }
).UseDatabase(connectionString:
              Configuration.GetConnectionString("firewall")
             , schema: "dbo"
             , dataRetention: TimeSpan.FromDays(365));
```

When you register your domain and obtain a license the registration process will have generated code samples showing the Links class as well as the minimal configuration options compatible with your license.

If your web application only hosts rest API then feel free to ignore the web service elements in the firewall options. There are quite a few more configuration options available to you, look at the online documentation here to learn more about `IFireWallConfig` interface.

> The default storage location of the firewall is on disk, however It is strongly recommended to make use of a database configuration of the firewall state using the extension methods.
>
> In the extension method you can configure the firewall to use shared databases for UserAgent and Internet Service Provider discovery and rules.

Securing the web application using the firewall

There are 2 ways to go about protecting your web application by using implicit security via the use of a Firewall filter on top of each controller or by enabling the Always-On feature of the firewall.

Always protect using Always-On

To configure Always-On protection you need to enable the FireWall middleware in your web application. Always-On protection is configured by adding **`app.UseFireWall()`** in the Configure section from Setup.cs, the below sample shows a simplified version of how your configuration might be. To enable advanced features like CSP and tamper detection use `UseSecurityHeadersMiddleware()`. More code samples on CSP are available here

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
   app.UseExceptionHandler("/Home/Error");
   app.UseHsts();
   app.UseStaticFiles();
   //option 2: this method also allows you to further configure the firewall
   app.UseFireWall().UseSecurityHeadersMiddleware();
   app.UseRouting();
   app.UseAuthentication();
   app.UseAuthorization();
   app.UseEndpoints(endpoints =>
   {
       endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");
       endpoints.MapRazorPages();
    });
}
```

Note that the firewall is set after `app.UseStaticFiles()`, this allows the firewall to protect static files as well as any endpoints in your application.

Having enabled static file protection allows you to be able to reject ad-hoc file requests. Rejecting ad-hoc file requests will disallow users to download your static files when the request is not based on a page served by your web application. This method prevents stealing videos, javascript, CSS files from your domain by search engines or users. More on this feature is available here. You can unprotect all static files by setting AlwaysOffOnStaticFiles to true.

If you get blocked during development you will need to reset the firewall’s incident data as well as delete the cookies in your browser. If licensed, you may need to delete any blocked rules in your server’s firewall (licensed add-on)

You can reset the firewall’s incidents by setting the value of **IFrireWall.Configuration.ResetFirewallViolationsOnEachStart** to **true**.

One word of caution when applying a reset on each start as the firewall will restart when your application restarts. Therefore, an attacker can reset his own incidents if he knows how to crash your application...

Protecting specific endpoints

You do not have to enable Always-On, you can also protect individual controllers or actions by the use of firewall annotation filter attribute as shown in the sample code below.

```
[Walter.Web.FireWall.Filters.FireWall]
public class HomeController: Controller
{
...
}
```

Protecting only single endpoints is a valid use case. However, using single page protection will make it harder for the firewall to protect against automated penetration attempts on resources that may not exist and because they do not exist one can know that the user is malicious.

Securing your endpoints

This chapter demonstrates how one can secure an endpoint with annotations. By default, the firewall secures your endpoints via its default rules in the configuration. You can override default protection and rules by defining rules on your endpoints using annotations. All annotations are located in namespace `Walter.Web.FireWall.Annotations` and can be combined, the order in which you combine your annotations is irrelevant. The online documentation for the annotations is a good reference to help you get the most out of the feature.

The BlockDuration Attribute

This attribute is not filtering any requests, it allows you to specify how long something should be blocked if a violation on a given endpoint is detected. The BlockDuration annotation can be applied on the controller, making it applicable to all actions in it, as well as apply on it on each specific action. If applied to the controller and action the action will override any assigned values.

If no BlockDuration is applied then the configured runtime value of `IFireWall.Configuration.Rules.BlockRequest.BlockDuration` will be applied by default. More on the BlockRequest configuration class is available here.

During development, one would ideally use a low value to test the blocking features without halting debugging your web application development. The bellow sample shows how you can use the BlockDuration attribute

```
[BlockDuration(seconds: 60,sliding:true,doubleDurationPerIncident:true)]
public IActionResult Index()
{
   return View();
}
```

The GeoBlock Attribute

By using the geographical attribute you can enable or disable controllers or actions when users come from a specific region or country. You can block or redirect a single or several locations at the same time.

To enable Geographical features you need to download one of the Walter.Web.FireWall.Geo.XXX NuGet packages and instructs the firewall to use them as shown in the samples below. Having defined that you intend to make your application geographical aware you can set specific exclusion and overriding rules as shown in the next 2 implementation steps.

Step 1 is to update your service configuration by including.UseGeography() on the AddFireWall extension method as shown below[1].

```
services.AddFireWall("Your Token", "Domain Key"
          , new Uri("https://www.MyDomain.com", UriKind.Absolute),
options =>{
  //your options
}).UseGeography();
```

Step 2, is optional and allows you to specify the default blocking configurations. The below sample shows how to disallow access from North Korea to all but a fixed IP address in North Korea, it's marked as being an office location.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
   app.UseFireWall().AddGeoBlockingMiddleware(options=> {
      options.Block(geoLocations: new[] {
                                Walter.BOM.Geo.GeoLocation.NorthKorea});
      options.Exclude("175.45.176.15",
                  Walter.Web.FireWall.Geo.GeoFreeAccessReason.Office);
      });
}
```

[1] Depending on the Walter.Web.FireWall.Geo.XXX NuGet package used you may have several configuration options in the UseGeography() extension method.

More information on how to configure geography can be found in the online help for each NuGet package. More on the MaxMind NuGet configuration can be found in the plugin section of this document or online here.

You can use the GeoBlock attribute to override controller or action to accept or block different geographical regions to that what you have defined in Startup.cs as shown in this sample.

```
using Walter.BOM.Geo;
using Walter.Web.FireWall;
using Walter.Web.FireWall.Annotations;
using Walter.Web.FireWall.Filters;
using Walter.Web.FireWall.Geo;

[GeoBlock(blockLocation: new[] { GeoLocation.AFRICA,
                              GeoLocation.LATIN_AMERICA,
                              GeoLocation.Netherlands }
      , redirectToController: "home", redirectToAction: "blocked")]
public class HomeController : Controller
{
  private readonly IFireWall _fireWall;
  private readonly IPageRequest _page;
  private readonly ILatLongRepository _latLongRepository;

  public HomeController(IFireWall fireWall
                    , IPageRequest page
                    , ILatLongRepository latLongRepository)
  {
    _page = page;
    _fireWall = fireWall;
    _latLongRepository = latLongRepository;
  }

  public async Task<IActionResult> Index()
  {
     var model= new HomePageViewModel();
     model.UserQuery = new MyIndexViewModel();
     model.WhoIsISP = await _fireWall.WhoisAsync(_page).ConfigureAwait(false);
     model.WhereAreYou = _latLongRepository.QueryMapLocation(_page.IPAddress);
     return View(model);
  }
```

code continues on the next page

```
    [GeoIgnore(maxRequest: 5)]
    [Ignore(skip: FireWallGuardActions.RejectPenetrationAttempts
                        | FireWallGuardActions.RejectRepeatViolations
                        | FireWallGuardActions.RejectWrongUserType
                  , skipCount: 5)]
    public IActionResult Blocked(BlockingReason id)
    {
        //the path home/blocked is defined as redirect on firewall rules
        var user= _page.User.AsFirewallUser();
        if (user.UserType.HasFlag(UserTypes.IsMalicious))
        {
            return View("_Malicious",id);
        }

        return View(id);
    }

}
```

The GeoIgnore Attribute

Instructs the firewall to ignore a geographically blocked request. Basically, to give the requester a pass on the geography filter that would cause the request to get an otherwise blocked response. The GeoIgnore attribute allows you to specify how often the requester is allowed to access the resource before being blocked. The annotation implies that you installed a GEO add-on like Walter.Web.FireWall.Geo.Native or Walter.Web.FireWall.Geo.MaxMind.

The ISPTrustLevel Attribute

You can configure the firewall to reject all data from an internet service provider based on the trust level you assign an IP address range that ICAN has assigned to the internet provider. This configuration allows you to grant or reject access to your application based on the owning service provider effectively configuring the firewall to block or allow cloud access to providers like AZURE, AWS etc if the consumers of your web application do not align with your target consumers. For example, no human user will visit your site from the AWS cloud, no REST Service for your corporate network will come from a data centre that is not hosting your services….

Internet service provider trust (ISPTrustLevel) can be configured as a default rule on the firewall configuration making it applicable to your whole site as well as an annotation on the individual endpoints.

There are 2 ways to alter the trust level, in code using the API or by using the FireWall Management application.

The bellow sample shows the firewall event subscription of OnIncident that will be triggered before an incident is generated

```
private void FireWall_OnIncident(object sender,
                                 FireWallIncidentEventArgs e)
{
    //if not already blocked because the ISP is blocked
    if (e.StackEntry.RuleNr != (int)RuleNumber.RejectAccessForISP)
    {
        //get the ISP
        var isp = e.Page.GetISP();

        //if not already black-listed
        if (isp.TrustLevel != ISPTrustLevels.BlockFilter)
        {
            //if more than 50% of requests are blocked
            if (isp.UsersWithBlockRate > 0.50M
                && isp.UsersWithBlockStatus > 10)
            {

                //then blacklist the ISP IP Range that this request
                // belongs to
                isp.TrustLevel =ISPTrustLevels.BlockFilter;

            }

        }
    }
}
```

You can “white-list” an ISP if for example, you would like to allow any Azure hosted request to be allowed if you are not sure where in Azure your company is hosting its services. Please note that one can configure the WHOIS source to be used by all firewalls in your organisation, a change will propagate to all applications that use the same source allowing you to alter the behaviour of another application without knowing.

You can use the Firewall desktop to see web applications that sources the ISP settings from a shared datasource. More information on the desktop can be found here

The Ignore Attribute

Instructs the firewall to ignore a subset of firewall violations, this can be "ALL" violations or as individual rules, as shown in the above sample.

Like GeoIgnore this also allows for leniency of the firewall rules allowing for a limited amount of access before blocking the request from a particular consumer.

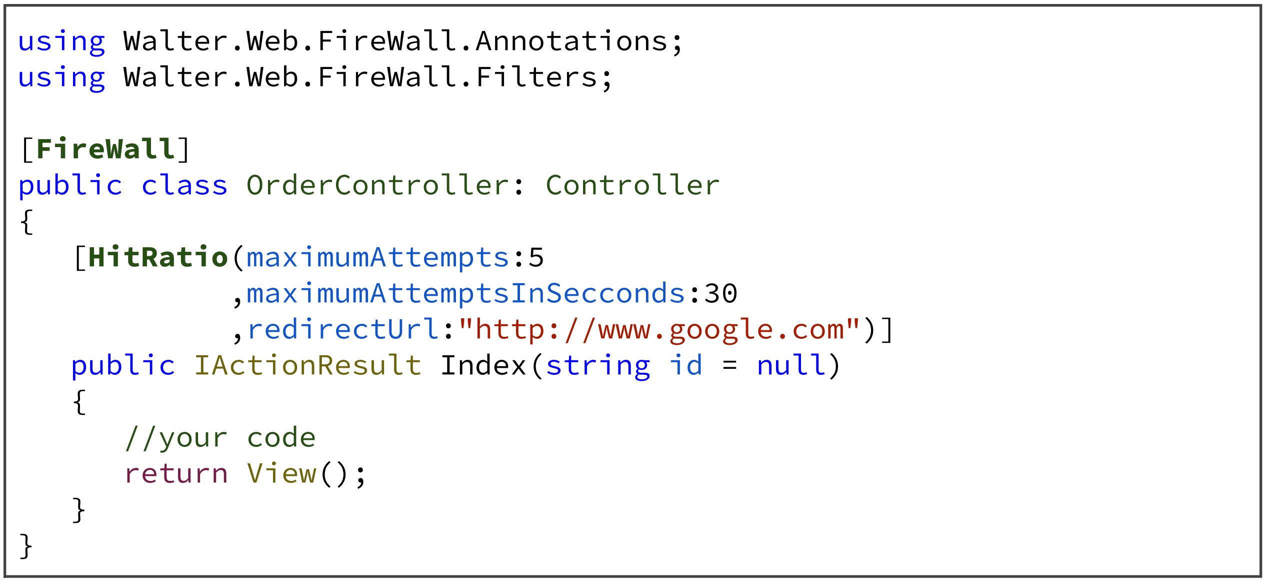
You can't turn off all rules in the firewall using the Ignore attribute. You can only disable the rules that are specified in FireWallGuardModules

The HitRatio Attribute

Being able to manage how often requests can hit a given page by a given user. If specified, the HitRatio allows for the users to be redirected to an external URL as well as redirect the user to a given controller, action and parameter as shown in the below sample.

An example. A user logging in with the right username and password for several users is either a “fake profile bot” or someone or something that has gained access using a compromised password database.

Note that user classification is quite extensive as users can be humans however users are not limited to humans and can also be bots, search engines, ISP’s, etc. more on users later in the UsersAttribute.



```
using Walter.Web.FireWall.Annotations;
using Walter.Web.FireWall.Filters;

[FireWall]
public class OrderController: Controller
{
   [HitRatio(maximumAttempts:5
            ,maximumAttemptsInSecconds:30
            ,redirectUrl:"http://www.google.com")]
   public IActionResult Index(string id = null)
   {
      //your code
      return View();
   }
}
```

The sample uses FireWall annotation on the class, this is not needed when Always-On is configured in Startup.cs, view sample page 9 for a sample of the Always-On configuration.

Firewall defaults that affect how non-annotated rules are executed in regards to the above properties. More detailed information on the rules class can be found here

The Users Attribute

The firewall tries to discover what type of user is as early as possible and will remember the user any later visits. 3 Basic user groups are never combined. The 3 main categories are *IsHuman*, *IsBotand and IsSearchEngine*.

On top of that, flags are indicating *IsIsp*, *IsMalicious* and finally *IsUsingDeveloperTools*. When using the User attribute you're able to allow and deny access to an endpoint based on the user assigned flags. A user could be Human, Malicious as well as a Bot can be Malicious; more detailed information on the user types here. You can access the discovered user by injecting the IPageRequest interface in your controller's constructor.

The path to the user type is IPageRequest.User.AsFirewallUser().UserType. You have access to the discovered type of search engine when accessing `IPageRequest.User.IsSearchEngine` property more on IsSearchEngine in the online documentation can be found here. The below sample demonstrates the fact that only humans are allowed to log in; all others are redirected to an action called captcha.

```
[Users( allow: UserTypes.IsHuman
       ,blocked: UserTypes.IsMalicious
       ,redirectToController: "home"
       ,redirectAction: "captcha")]
public IActionResult Login()
{
   return View(new LoginModel());
}
```

If the user type is known at the time of the request 2 things can happen, the user is allowed in as he has a flag stating he IsHuman, or he is redirected to the specified action that will either validate the user type or take other actions.

Let's look at a sample implementation of working with the `IFireWallUser` as well as `IPageRequest` and the `IFireWall` interfaces in a simplified captcha action. First, let's set up the controller to allow access to these interfaces using the following rather long section of sample code starting on the next page.

```
using Walter.Web.FireWall;
using Walter.Web.FireWall.Annotations;
using Walter.Web.FireWall.Filters;
using Walter.Web.FireWall.RuleEngine.Rules;

[BlockDuration(seconds: 60, sliding: true)]
public class HomeController : Controller
{
   private readonly IFireWall _fireWall;
   private readonly IPageRequest _page;


   public HomeController(IFireWall fireWall, IPageRequest page)
   {
      _page = page;
      _fireWall = fireWall;
   }

   [HttpGet]
   [Ignore(skip: FireWallGuardActions.RejectWrongUserType)]
   public IActionResult Captcha()
   {
      //1. Test if the url that was used to get to us came from
      //   this web application
      var previous = _page.Referer?.IsAbsoluteUri.ToString()
                           ?? _page.User.PreviousPage;

      if(!Uri.TryCreate(previous, UriKind.Absolute, out var url)
          && !_fireWall.License.IsLicensedDomain(url))
      {
          //should have a valid referrer page therefor leave
          return RedirectToAction(actionName:"Index",
                                  controllerName:"home");
      }

       //2. All ways block hackers user or not
      var user= _page.User.AsFirewallUser();
      if ( user.UserType.HasFlag(UserTypes.IsMalicious)
        || user.UserType.HasFlag(UserTypes.IsUsingDeveloperTools))
      {
          return RedirectToActionPermanent("Blocked");
      }
```

code continues on the next page

```
 //3. Bot's are always blocked as captcha assumes human activity
 if( user.UserType.HasFlag(UserTypes.IsBot) )
 {
    //permanently block this user for 30 days and
    //flag it for reporting
     _page.Block(UserBlockingRule.WrongUserType
                         , $"{user} rejected from {previous}"
                         , TimeSpan.FromDays(30));
  }

  //4. Generate a page that forces user interaction and remember
  ///  the referrer as well as previous submitted invald models
  return View(new CaptchaModel(redirectTo: previous
                             ,errorCount: user.ModelBlockCount));
}

[HttpPost]
[Ignore(skip: FireWallGuardActions.RejectWrongUserType)]
[ActionProtector( blockAfterInvalidModelCount: 5
             , redirectToController: "home"
             , redirectToAction: "blocked"
             , passModel: false)]
[ValidateAntiForgeryToken]
public IActionResult Captcha(CaptchaModel model)
{
   var user = _page.User.AsFirewallUser();
   //1. Make sure the page came from within the web application
  if( _page.Referer is null
     || !_fireWall.License.IsLicensedDomain(_page.Referer)
    || !_page.Referer.AbsolutePath.Contains("captcha",
                StringComparison.OrdinalIgnoreCase)
    )
  {
    //if was triggered by a search engine then ignore it else
    //flag it for later protection when revisited are discovered
     if(_page.User.IsSearchEngine != SearchEngine.NotSure)
     {
           user.UserType |=   UserTypes.IsMalicious
                            | UserTypes.IsUsingDeveloperTools;
     }
  }
```

code continues on the next page

```
   // 2. Make sure the user passed the simple test if failed then
   //  assume that if “ the user” failed 3x that he is up to no good.
   if(!ModelState.IsValid)
   {
      if(user.ModelBlockCount - model.ErrorCount > 3)
      {
         user.UserType |= UserTypes.IsMalicious;
      }
      return View(model);
   }

   // 3. Reject and send user to the blocked action if we
   //    discovered spoofing or other unwanted activities.
   if(_page.User.IsSpoofing()
      || user.UserType.HasFlag(UserTypes.IsMalicious)
      || user.UserType.HasFlag(UserTypes.IsUsingDeveloperTools))
   {
      return RedirectToActionPermanent("Blocked");
   }

   // 4. Process the model and send the user to its final
   //    destination.
   switch (user.UserType)
   {
      case UserTypes.IsHuman:
      case UserTypes.IsSearchEngine:
           return Redirect(model.RedirectTo);
      case UserTypes.NotDiscovered:
           return View(model);
      default:
           return RedirectToAction("Blocked");
    }
}
```

The sample does not contain the code for the CaptchaModel class, the only need for this class is to store the redirect to logic, and the model validation error count. The user interaction can be generated with google's CAPTCHA script or simply a drop-down list and checkbox that causes the user to interact with the page.

User discovery, as well as nice web statistics, are generated by the firewall web services configuration and classes. The discovery framework uses a javascript that will be generated by the firewall on each request and is adapted to each user. This javascript then calls pre-defined endpoints in your application or default web services inside the firewall to process discovery data.

When you registered your firewall license the License wizard generated the source code for your application to use and will therefore not be covered in this manual.

The WhiteListIP Attribute

Sometimes endpoints need to be protected so that only a single IP address can access them. You will find that protecting an endpoint with a username and password is not always sufficient and being able to explicitly allow a given IP address, or Range of IP addresses will add that extra protection needed for mitigating against ex-employees as well as hackers.

```
[HttpGet]
[Authorize(Roles = "Accounting")]
[WhiteListIP(Constants.OfficeIPAddress)]
[Users(users:UserTypes.Anyone,areAllowed:false)]
public IActionResult CreditCardPayments()
{
   return View();
}

[HttpGet]
[Users(users: UserTypes.IsSearchEngine)],
[WhiteListIP("8.8.8.2", "8.8.8.8")]
public ContentResult GoogleSiteMap()
{
   var xml = _fireWall.KnownLinks.GetGoogleSitemap();
   return new ContentResult()
   {
       Content = xml,
       ContentType = "application/xml",
       StatusCode = 200
   };
}
```

The previous sample will block all access by all types of users. The firewall will not test the configuration from the users' annotation if the IP is whitelisted. Those that attempt to access the endpoint from another location will be blocked.

You could easily test for the IP address yourself, however blocking like this will, depending on your configuration, do far more than simply returning an empty view or a redirect as it integrates into the firewall's reporting infrastructure. You can also use the WhiteListIP on controllers protecting the whole controller in one step. You can also disable whitelisting by setting `IFireWall.Configuration.Rules.AllowWhiteListing` to `false`

The WhiteListSubnet Attribute

Sometimes in corporations, you have a website that serves both internal and external users. If these users are allowed to access links that are not available when not connected to your network (can be users in-home-office connected via VPN) then the WhiteListSubnet attribute will be ideal for your use case.

WhitelistingSubnet works on all actions in a controller or can be specified on a single action, no arguments are required as it will take the IP address of the server and allow access to anyone in the same subnet of the server.

```
[WhiteListSubnet]
public IActionResult HumanResources()
{
      return View();
}
```

The ActionProtector Attribute

Having the possibility to deal with repetitive false submissions on an endpoint is not only annoying but can indicate that among other reasons you're exposed to:

1. A malicious human
2. A bot that's trying to brute force access to data
3. An automated agent trying to generate entries in your database.

You have seen the attribute in use on the above CAPTCHA sample, more information on the ActionProtector[2] attribute is available here.

Sometimes you like a little bit more control over the firewall guard actions on your endpoints. If you annotate your firewall with an ActionProtector attribute then you can change the protective action on the page as you can override the IPageRequest GuardAction as well as TransferedTo properties that you're able to alter in the action's scope.
This will add some compute power that could be saved as your action is executed but gives you insight into the status of the request as well as the violations it recorded.

The RedirectRule Attribute

This attribute allows for configuring the redirect portion on all annotations that have a redirect property configured. The below code allows for 1 redirect in any given hour.

```
[RedirectRule(maxRedirects: 1, timeSpanInSeconds:3600 )]
[Users(Walter.Web.FireWall.UserTypes.IsSearchEngine,"home","index")]
public IActionResult OnlySearchEngines()
{
   return View();
}
```

The redirect count and the increments are stored on the user, not on the endpoint, it is important to understand that if the user consiômes several different endpoints and get redirected he will increment his redirect count to a point where the user will be blocked, not blocked on a given endpoint, blocked on the whole web application.

If the redirect rule is defined the no incident will be created when a rule is triggered. The default TimeSpan in TolerateMaximumViolationsIn and the count specified in TolerateMaximumViolations do not specify that no incident is generated, they merely specify that the user is not blocked. The RedirectRule attribute is a form of Ignore attribute where it relaxes the firewall rules under a specific condition.

[2] This protection method Is available on all registered firewall licenses, also in free versions.

The CrossSite Attribute

Sometimes an endpoint is available only when the endpoint is communicated from within a trusted domain. You can use XSS and CORS protection in browsers by configuring this in header rules, this, however, does not protect you when a user is not using a modern browser.

This vulnerability is closed by the firewall when you are using the CrossSite attribute on your endpoints as shown in the below sample.

```
[HttpPost, Route(Links.SiteMapEndPoint)]
[CrossSite(useDefaultRedirect:false)]
[Ignore(skip: FireWallGuardActions.ALL
              &~FireWallGuardActions.RejectCrossSiteRequests)]
public StatusCodeResult SiteMap([FromBody] SiteMapDiscovery model)
{
    if (model is null)
        return NoContent();
    else
    {
        // The firewall will use predictive navigation for user as
        // well as use it in web-statistics to show choices made
        // by the user helping the marketing department
        _fireWall.LogSiteMap(_page, model);
        return Ok();
    }
}
```

The above method informs the firewall to Ignore all rules but the RejectCrossSiteRequests rule. A general rule of thumb is **if you have a HttpPost, HttpPut or HttpDelete then add the CrossSite attribute** to prevent malicious manipulating of your data. Please note that authorisation attribute does not protect unwanted external manipulation.

You can allow external domains to interact with your endpoints by including them in the CrossSite attribute or by defining external trusted sites in the `TrustedCrossSiteDomains` property of the `IFireWall.Configuration.Rules` class.

Global protection is common in use-cases where you have API endpoints that serve a corporate website like a registration or payment gateway. By using the configuration of the firewall rules you specify them only once.

The Disabled Attribute

You can disable the firewall protection on any controller or action. What will happen is that no checks will be performed and the request will always execute. This however does not prevent you from accessing the IPageRequest object that will be generated by the system if you chose to include it in your controller’s constructor.

```
[Route("api/[controller]")]
[ApiController]
[DisableFirewall]
public class ServicesController : ControllerBase
{
    // GET: api/Services
    [HttpGet]
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
     }
}
```

Using browser-based protection

Having briefly mentioned the `UseSecurityHeadersMiddleware()` method to enable “always-on” protection rules on the headers of your web requests. Having it to be “always-on” is not always desired.

There are 2 ways to configure browser protection, one way is the aforementioned `UseSecurityHeadersMiddleware()` the other way is simply in the firewall configuration. Both methods have the same capabilities and provide the same configuration in the browser.

The bellow code will not enable always on for the firewall but will enable always-on on each request for CSP and XSS protection

```
app.UseSecurityHeadersMiddleware()
                .AddDefaultSecurePolicy()
                .AddStrictTransportSecurityNoCache()
                .AddXssProtectionBlockAndReport()
                .AddContentSecurityPolicyButTrust(
                       trustingSites: |TrustingSites.Self
                                     |TrustingSites.Jquery
                                     |TrustingSites.Google
                      , allowInline: true
                      , framesPolicy:FramesPolicy.Self);
```

The below code will enable the exact same browser protection, whenever the firewall protects an endpoint only without the configuration endpoint in the middleware.

```
services.AddFireWall("Your Token", "Domain Key",
       domainName: new Uri("https://www.test.dll", UriKind.Absolute),
       options => {
                options.WebServices.CSPReportUrl =
                     new Uri(Links.CSPViolation, UriKind.Relative);
                options.Rules.Headers.AddDefaultSecurePolicy();
});
```

You can get more information on the individual methods in the header class as well as a large sample online using this link.

Inspecting and manual altering the endpoint protection

The firewall will generate a new configuration file for the discovered endpoints in the Data folder specified in the IFireWall.Configuration.DataFolder property. The file is in JSON format and will be read if the file is newer than your web application. You can prevent the firewall from using the file by disabling it by setting `LoadEndpointConfigurationFromDisk` to false in `IFireWall.Configuration.Rules` configuration class

You may choose to set this value to false if you are on a hosted server as your security may get compromised if the server gets compromised.

You can access all endpoints known to the firewall by accessing the `IFireWall.KnownLinks.Endpoints property.`

> If you have not licensed the *Walter.Web.FireWall.SqlLogger* NuGet package then you do not have access to a SQL Server Database and advanced reporting will not be possible. If you only need simple web statistics then you can use Linq to generate your reports from the Endpoints as it collects resource usage statistics. Have a look at the DiscoveredRout class and its properties.

You can disable Static file protection on all static files by setting AlwaysOffOnStaticFiles in Configuration.Rules to true, or disable only specific endpoints by subscribing to the IFireWall.Configuration event OndEndpointsCreated as shown in this sample.

```
private void Options_OnEndpointsCreated(IKnownLinks links)
{
    foreach (var item in links.EndpointsInPath("*.css", "*.png"))
    {
        item.FirewallDisabled = true;
    }
}
```

Reporting on the health of the firewall and viewing the activity

The firewall contains a lot of data that can be helpful when it comes to reporting on its overall state as well as the integration into your project and one of the easiest ways to access this data is via the reporting API of the IFireWall.

These methods allow for plain text as well as strongly typed output where the text is made for humans and does not contain all data. The strongly typed data is provided by the ReportingData class for the MetaData and the details in the form of nested class ReportingData.ReportDetails.

> Please note that firewall reports can contain security-relevant data, it is therefore recommended to guard it with a password or limit access using one of the IP address related annotations like WhiteListIP and WhiteListSubnet.

The below sample shows you how to call the firewall's reporting engine in your project protected by an authorization filter. It also tells the firewall to disable rules for the controller.

```
[Ignore]
[Authorize]
public class HealthController: Controller
{
  private readonly IFireWall _fireWall;
  public HealthController(IFireWall fireWall)
  {
    _fireWall = fireWall;
  }

  [HttpGet]
  [Produces("text/plain")]
  public string Index()
  {
    return _fireWall.Report(ReportTypes.DEFAULT);
  }

  [HttpGet]
  public JsonResult Json()
  {
     return Json(_fireWall.GetReport(ReportTypes.DEFAULT));
  }
}
```

The firewall can produce reports in several formats. Available formats are TextPlain and JSON. The easiest way to access JSON is shown above. This uses the ReportingData class and is accessed via the `IFireWall.GetReport(ReportTypes)` method. This way you do not have to escape the JSON if used in a web service as shown above.

> 💡 If you're having exceptions in your health report then you should with the customer improvement program. Being registered will automatically send any framework exceptions you have and any fix notifications will be included in the firewall and SMTP report. We will not email you as we store the hash of the email and not the email itself. The email hash in combination with your license allows us to identify and send your firewall update notification via the data pull subscription.

What to do when the firewall is blocking and you are not sure why

There are several reasons why a false positive could occur. The 3 most common reasons are:

1: You are using link names that are commonly used when hacking an application. You always need to test your links, access to resources such as Backup etc will trigger red flags.

2: You are limiting access on an endpoint to a user type before having discovered the user type. If interactive, then redirect the user to a specific action where you can mitigate or inform the user of the state.

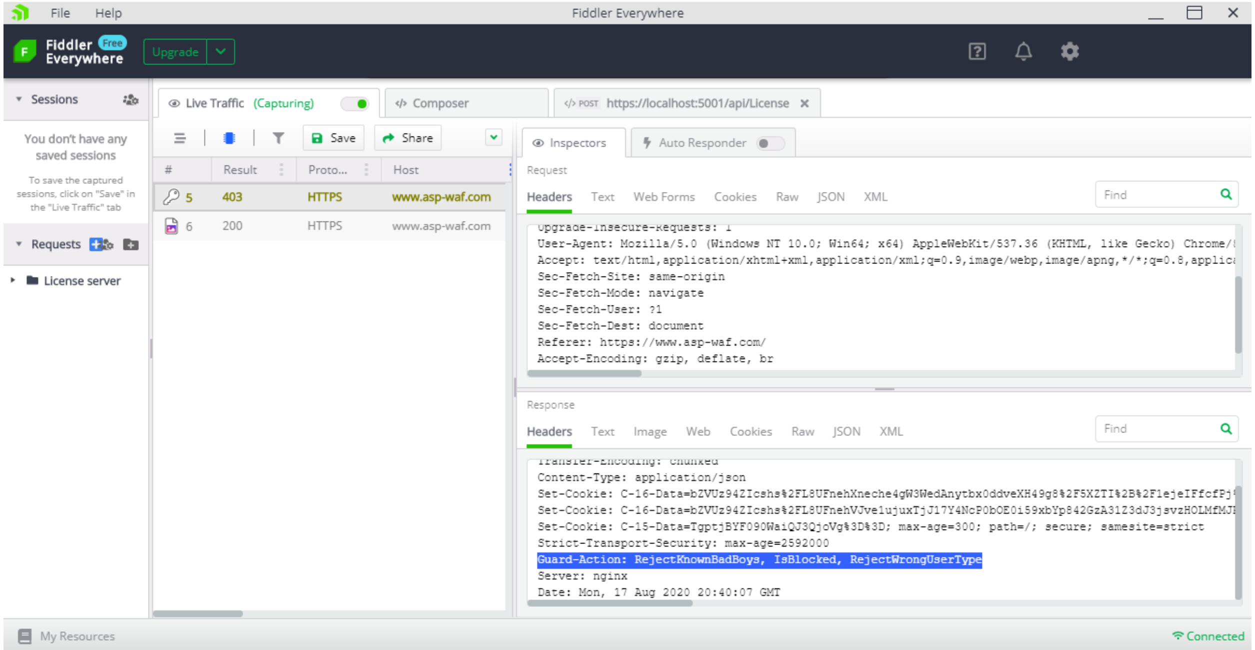
3: You are getting blocked because a javascript generates too many push or pull requests triggering a scrubbing or DOS incident. Depending on your configuration you could get redirected away from your page after the page has loaded while interacting with a page.

On remote server

You can use the Plugin Walter.Web.FireWall.EventLog to have the firewall log all incidents in an easily acceptable location without compromising security.

During development

If you are debugging your application then the firewall will write a header entry named Guard-Action that you can look at to see the rule that triggered the reason the HTTP request is blocked or redirected, to view the headers of your request you can use the F12 developer tools of your browser or use proprietary tools like Fiddler.



#	Result	Proto...	Host
5	403	HTTPS	www.asp-waf.com
6	200	HTTPS	www.asp-waf.com

```
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,applica
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://www.asp-waf.com/
Accept-Encoding: gzip, deflate, br
```

```
Transfer-Encoding: chunked
Content-Type: application/json
Set-Cookie: C-16-Data=bZVUz94ZIcshs%2FL8UFnehXneche4gW3WedAnytbx0ddveXH49g8%2F5XZTI%2B%2FlejeIFfcfPj
Set-Cookie: C-16-Data=bZVUz94ZIcshs%2FL8UFnehVJve1ujuxTjJ17Y4NcP0bOE0i59xbYp842GzA31Z3dJ3jsvzHOLMfMJ
Set-Cookie: C-15-Data=TgptjBYF090WaiQJ3QjoVg%3D%3D; max-age=300; path=/; secure; samesite=strict
Strict-Transport-Security: max-age=2592000
Guard-Action: RejectKnownBadBoys, IsBlocked, RejectWrongUserType
Server: nginx
Date: Mon, 17 Aug 2020 20:40:07 GMT
```

A blocked request will have the status code that will be returned to the browser can be provided during the initialization using the Rules.BlockRequest.StatusCode property, the default is 403 (forbidden).

During integration testing you might set Rules.Headers.ShowGuardActionsInHeader to true to see reasons why the firewall may reject your requests. If the set the property Rules.BlockRequest.ShowBlockingRule, to true, then the blocking rules will be shown instead of a blank page.

You can always generate Guard-Action in the response header by setting the value of `ShowGuardActionsInHeader` to true in `IFireWall.Configuration.Rules.Headers.` More on headers here

Doing so will show potential malicious users that you are using the firewall as well as the rules that they have triggered giving them clues of how to penetrate your application.

The default is on when a debugger is attached to the process, the values are false when not.

Getting Blocked by your own BlockedPatterns

The firewall inspects the HttpRequest object and tries to identify those URLs that are used by PEN scripts or those used by malicious users crafting URLs to submit using tools like WireShark, PostMan Fidler etc.

The firewall on starting will generate endpoints that are to be protected using the Actions and Pages that it discovers in your application. After these endpoints are discovered (or loaded) the firewall will validate if the current blocking patterns are in violation with your endpoints and mark the endpoints that would cause a false positive as containing errors an update the NoValidate property to disable the firewall guard module RejectPenetrationAttempts.

You will be informed when this happens by

1. ILogger formats and writes a critical log message using the following template: *"{endpoint} updated by the firewall as you would block yourself. Change the url or update Configuration.Rules.BlockedPatterns.{val} to allow for {pattern} on {location}".*
2. The Firewall will generate a ToDo item of type ConfigurationTasks telling you to update a blocking pattern to allow a URL.

3. The firewall will mark the endpoint with an entry in the `DiscoveredRout.BuilderErrors` and set the `DiscoveredRout.NoValidate` property to include `FireWallGuardActions.RejectPenetrationAttempts`

You can follow the recommendation but actually, you have created an application tailored for known exploits. What you should do is understand the routes that are affected and alter your application to not use these URLs.

Typical mistakes are using framework defaults like default login in the .net framework and change the paths and block the default scaffolding values of */Identity/Account*

```
services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = "MyAccount/UserLogin";
    options.AccessDeniedPath ="MyAccount/UserAccessDenied";
});
```

When changing the defaults make sure you test the scaffolded URLs and razor page URLs as well as other service configurations as they might depend on each other.

> 💡 When developing your application you should always consider using the firewall events. Firewall events allow you to look into the “black box” and see why some requests are being rejected while you are debugging.

Taking control over the firewall

You have the ability to use the firewall as a black-box and get started as shown in this document. There is nothing wrong with that approach however the real power comes with the flexibility to inject your own FireWall instance where you fine-tune the rules and well as integrate any logic you are currently missing in our security strategy.

Your own IFireWall implementation by inheriting from the FireWallBase class

To use your own IFireWall instance you have to make a class and base it on the FireWallBase class and use the type when registering your typed firewall when configuring your application services.

```
services.AddFireWall<MyFireWall>(FireWallTrial.License
     , FireWallTrial.DomainKey
     , domainName: new Uri("https://www.test.dll", UriKind.Absolute)
     , options =>{
      // your options
});
```

There are several methods in the FireWallBase class that you can use to override. However, the most control comes from injecting your own code on the fireWall events `OnResourceRequested`, `OnIncident` and `OnGuardAction`, more on those later and in the online documentation.

```
public class MyFireWall:FireWallBase
{
   public MyFireWall(ILoggerFactory factory,IMemoryCache cache)
            :base(loggerFactory:factory, memoryCache:cache)
   {
        base.Trigger_OnFireWallCreated(this);
   }
}
```

The above-oversimplified constructor will work fine, you can provide the other optional interfaces and let the dependency injection populate them for you, or create and inject your own.

Events on the IFireWall interface and FireWallBase class

Events are quite powerful and allow developers that are using the firewall framework to dive-in and add a point where you can interact with the firewall and start custom workflows as well as generate their own rules and pre- and post-processing workflows.

> The firewall will only invoke events if you are actively debugging or have a licensed instance of the firewall. Unlicensed firewalls, or firewalls that are not active due to a missing payment, will be silent and not invoke events.

The OnResourceRequested event

This event will be invoked for each resource that is managed by the application. The event is triggered on anything you can request using a Url or load in a page if you have specified the use of static files in your service configuration. The below sample shows how to use it for a simple country counter.

```
public class MyFireWall:FireWallBase
{
     readonly IDictionary<GeoLocation?, int> _access =
                          new ConcurrentDictionary<GeoLocation?, int>();

     public MyFireWall(ILoggerFactory factory
                     , IMemoryCache cache
                     , ILatLongRepository geo)
     : base(loggerFactory: factory, memoryCache: cache,
             latLongRepository:geo)
     {
          base.Trigger_OnFireWallCreated(this);
          OnResourceRequested += FireWall_OnRR;
     }

   private void FireWall_OnRR(object sender,PageCreatedEventArgs e)
   {
       if (_access.TryGetValue(e.Request.Country, out var counter))
```

```
        _access[e.Request.Country] = counter + 1;
      else
        _access[e.Request.Country] = 1;
    }
}
```

The Page Created event is triggered before the page is sent to a filter, middleware or an action. You are accessing a raw object and make sure you understand the implications on changes you make on any property.

The page is communicating with the reporting endpoints automatically after the request has been sent to the requestor for statistics processing. More on the `PageCreatedEventArgs` class is available online.

The OnResourceSend event

Being able to post-process a request can be quite handy at times. The `OnResourceSend` event allows you to do that. The `OnResourceSend` contains the `IPageRequest` that was sent to the client and contains the state of the request context after all processing and the IPageRequest has been sent to all reporting instances like SMTP, DiskLogging, SqlLogging, EventLogLogging etc.

The framework allows you to document any activity using the `AddDocumentation` method on the `IPageRequest` interface. You can use this method in combination with the `OnResourceSend` event, or the `OnRootPageRequestDisposed` event to understand and document code flow in your application.

The OnRootPageRequestDisposed event

This event is fired when a root page is released from memory. This method is like OnResourceSend however triggers just before being garbage collected by dot net. This method is similar to the `OnResourceSend` event however this event is only triggered if the request is considered to be a RootPage.

This method guarantees that the IPageRequest has been processed by all internal systems like firewall session management, storage and reporting interfaces. This event is guaranteed to execute nor is it real-time as there is no way to know when the garbage collector releases the memory.

These types of events are quite useful as you can see on the next page where we have added some sample code that is used to send emails to the developer when a page is processed that contains one or more exceptions. The sample assumes that you use the IPageRequest.AddDocumentation method similarly as what you would do with the ILogger interface adds “developer type” information that might assist in helping solving issues and bugs.

```
private void MyFireWall_OnResourceSend(object sender,
                                       PageSendEventArgs e)
{

   if (e.Request.Exception != null
        && _reported.Add(e.Request.OriginalUrl))
   {
      var sb = new StringBuilder();
      sb.AppendLine($"Page {e.Request.OriginalUrl} from
{e.Request.Country} caused {e.Request.Exceptions.Count} exceptions");

      foreach (var item in e.Request.Exceptions)
       {
           sb.AppendLine($"    {item.GetType().Name} exception
Message: {item.Message}");
           sb.AppendLine($"    {item.ClassName()}.{item.MethodName()}
in {item.FileName()}:{item.CodeLineNumber()}");
           sb.AppendLine(item.StackTrace);
           sb.AppendLine();
      }

       var docu = e.Request.GetDocumentation();
       if (docu.Count > 0)
       {
           sb.AppendLine("Documentation:");
           foreach (var item in docu)
           {
              sb.AppendLine(item.Message);
              sb.AppendLine($"{item.Method}
{item.FileName}:{item.Line}");
           }
       }
       //helper class to send emails to developers
       MailClient.SendDeveloperError(body: sb.ToString()
                                    ,subject: e.Request.ToString());
     }
}
```

Use events for advanced runtime configuration

The firewall configuration allows for targeted updates on the configuration settings after having been activated. Available are the following 3 events OnRulesCreated, OnEndpointsCreated and OnFireWallCreated.

The IFireWallConfig.OnRulesCreated event

You have the ability to alter When the firewall rules configuration has been created and before the RuleEngine is instantiated. This event is ideal to make changes to advanced configurations containing runtime rules like RuleConfig.BlockedPatterns. The following code show how to update the pattern to allow access to the default .net default membership path */Identity/Account/*

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
       , new Uri("https://www.domain.com",UriKind.Absolute)
       , options =>{
         options.OnRulesCreated+=(sender, rules)=> {
             rules.BlockedPatterns.NoPublicAccessToAdministration
                                  .Remove("/Identity/Account/")
         };
});
```

The OnEndpointsCreated event

The bellow sample shows how all items in the downloads folder as well as all *.zip and *.pdf can be accessed externally from links in documents like this without invoking a AddHockFileAccess violation .

```
options.OnEndpointsCreated+=(sender,args)=>  {
   foreach (var item in args.Links.EndpointsInPath( "Downloads/*",
                                                "*.zip", "*.pdf" ))
   {
      item.AllowAddhockFileAccess = true;
   }
};
```

The OnFireWallCreated event

You can create your own custom IFireWall class by inheriting from the FireWallBase class and subscribe to the IFireWall events, you, however, do not have to. You can use the OnFireWallCreated event during configuration to work with the firewall's default implementation.

You can access the default implementation during the configuration of the service by assigning the events in the configuration. If you load the firewall configuration from IConfiguration then this would not work and access to the IConfiguration events are not accessible.

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
    , new Uri("https://www.your-domain.com", UriKind.Absolute)
    , options =>{
       options.OnFireWallCreated+=(firewall)=> {
            //get the current configuration
            var json=  JsonConvert.SerializeObject(firewall.Configuration);
            //get filename in the root of the application
            var fileName = Path.Combine(Directory.GetCurrentDirectory()
                           ,"last known good firewall configuration.json");

            //tell firewall to write it to disk as uncontrolled changes
            // to disk will raise an incident
            firewall.WriteFile(new FileInfo(fileName), json);

            //create methods for me to set breakpoints on to
            //understand why incidents are created, requests are blocked
            //and user groups are assigned and or are changed
            firewall.OnGuardAction +=FireWall_OnGuardAction;
            firewall.OnUserTypeChange+=FireWall_OnUserTypeChange;
            firewall.OnIncident+FireWall_OnIncident;
      };
});

private void FireWall_OnIncident(object sender,FireWallIncidentEventArgs e)
{
#if DEBUG
    Debugger.Break();
#endif
}

...
```

Events allowing you to control the firewall’s recommendations

When you are new to the firewall you sometimes would like to test the firewall’s guard actions to ensure that they are not too restrictive. The `OnIncident`, `OnGuardAction` and `OnUserTypeChange` events allow you to among others log the recommendations as well as provide post-processing in your own code.

An extensive code sample of how to use the below events is shown in the FireWallBase constructor that you can access online from here.

The one thing these events all have in common is that they have an **Allow** property that will default to false meaning the firewall is not allowed to implement the recommendation. **If you register the event and do not explicitly set it to true the firewall will effectively become read-only.**

The FireWallBase.OnIncident event

The event is raised before an incident entry is generated on a specific rule with reasons property showing why a rule triggered. More information, as well as code samples on the `FireWallIncidentEventArgs` class, is available online.

You can use the event to adjust the default rules as well as your endpoint configuration. The following sample shows how such code could be implemented.

```
private void FireWall_OnIncident(object sender, FireWallIncidentEventArgs e)
{
    if (e.StackEntry.Reason == FirewallBlockReasons.PenetrationAttempt
       && Enum.TryParse<UrlValidatedResult>(e.Data[IncidentAssertType.RuleNr]
         , out var list))
    {
        _logger.LogCritical("Url {url} triggered {pattern} in {location} in
Configuration.Rules.blockedPatterns.{list}"
        ,e.Page
        , e.Data[IncidentAssertType.Pattern]
        , e.Data[IncidentAssertType.Location]
        , list);
    }
}
```

A more in-depth example is available online here for working with the BlockedPatterns property.

The FireWallBase.OnGuardAction event

If the firewall generates a “blocking” or “redirect” recommendation the firewall will invoke an `OnGuardAction` event allowing you to log the event as well as perform some post-processing on it using your code. More information on this event can be found online

```
private void MyFireWall_OnGuardAction(object? sender,
GuardActionEventArgs e)
{
   _logger?.LogCritical("{Method} {page} : {route}\n
{action}:{RuleNr}\n   Reasons:{Reason}\n   {data}"
    , e.Page.Method, e.Page.OriginalUrl.AbsolutePath
    , e.Page.FireWallRoute, e.Action
    , string.Join("\n   "
                , e.Page.ViolationsStack.Select(s => s.ToString())));

     //never allow the firewall to block requests
     e.Allow = false;
}
```

You can access all incidents that occurred on the page by accessing the IPageRequest.Incidents() methods. This does not contain the same level of detail that you have in the OnIncident event however it provides a fair amount of data that is used to evaluate future requests by the user on the firewall and determine if access is granted.

Another example is the altering of incidents that have been recorded and will affect future firewall evaluations. The following code shows how you can alter the duration of a block by changing the expiration date of the incidents in the case the user was also spoofing.

```
private void FireWall_OnGuardAction(object sender,
                                    GuardActionEventArgs e)
{
   if (e.Page.User.AsFirewallUser().IsSpoofing)
   {
       foreach (var item in e.Page.Incidents())
       {
          item.Expires = item.Expires.AddDays(30);
       }
   }
}
```

The FireWallBase.OnUserTypeChange event

This event is invoked when the firewall assigns the user to one or several user groups. You can alter the assigned values by changing the NewType property or ignore it by not setting Allow to true.

The below, oversimplified sample, shows how such an implementation will remove the IsMalicious flag from the user groups if the firewall detected an activity it did not like. More information on the OnUserTypeChange event can be found online here

```
public class MyFireWall: FireWallBase
{
     private readonly ILogger<MyFireWall> _logger;

     public MyFireWall(ILoggerFactory loggerFactory
                     , IMemoryCache memoryCache)
           : base(loggerFactory, memoryCache)
     {
          base.Trigger_OnFireWallCreated(this);
          OnUserTypeChange += MyFireWall_OnUserTC;
          _logger = loggerFactory.CreateLogger<MyFireWall>();
     }

     private void MyFireWall_OnUserTC(object? Sender
                                    ,UserTypeChangedEventArgs e)
     {
         _logger?.LogCritical("{oldType} : {newType}"
                              , e.OriginalType, e.NewType);
         //allow the change
         e.Allow = true;

          //if the user is a search engine then remove flags that
          //are to restrictive for this application
          if (e.OriginalType.HasFlag(UserTypes.IsSearchEngine)
                     && e.NewType.HasFlag(UserTypes.IsMalicious))
          {
             //remove the malicious flag from search engines to not
             //prevent search engines from indexing the site
             e.NewType &= ~UserTypes.IsMalicious;
          }
     }
more code of the class not relevant to this sample
```

Reset FireWall incidents

Having the ability to reset the firewall is an important feature as physical access to the firewall backing storage is not always practical or even possible and removing all incidents will cause all known offenders to be able to gain access again to the system. It's for that reason that you can remove specific geographical regions address ranges, date ranges or users[3].

The IFireWall.ResetFireWallBlocking method

You can reset the blocking features of the firewall by "expiring" incidents that have been recorded. There are several ways to do this, all are controlled via the method argument.

Perhaps one of the most important once is having the ability to email a registered user that has been locked out of the firewall. In this case, you would send them an email with a link that calls the ResetFireWallBlocking for that user.

```
public class RejectController : Controller
{
    private IPageRequest _page;
    private IFireWall _fireWall;
    public RejectController(IPageRequest page, IFireWall fireWall)
    {
        _page = page;
        _fireWall = fireWall;
    }

    [DisableFirewall]
    public IActionResult ResetBlockedUsers(string resetKey)
    {
       //test if the resetKey belongs to the user
       if (_page.User.GetUserSalt() == resetKey)
       {
          var user= _page.User.AsFirewallUser();
          _fireWall.ResetFireWallBlocking(user);
       }
       return RedirectToAction("index", "Home");
    }
}
```

[3] Removing the incidents will also invalidate any blocking cookies issued to the user.

The IFireWall.Clear method

The firewall collects quite a lot of data while it’s active and sometimes you just need to be able to remove certain items while leaving other data intact. The Clear method allows you to remove firewall data remotely from within your application. Have a look at the online documentation for the data types that you can remove. The below sample code shows how you would go about implementing this in your application.

```
[Authorize]
public class ManagementController : Controller
{
    private IPageRequest _page;
    private IFireWall _fireWall;
    public ManagementController(IPageRequest page, IFireWall fireWall)
    {
        _page = page;
        _fireWall = fireWall;
    }
    [CrossSite]
    public IActionResult Clear(string resetKey)
    {
         //test reset key valid
         _fireWall.Clear(ClearOptions.FireWallIncidents | ClearOptions.WhoisState);
        return RedirectToAction("index");
     }
     [CrossSite]
     public IActionResult ReportOnly()
     {
         if(_fireWall.State==FireWallState.Active
             ChangeStateTo(FireWallState.ReportOnly );
         Else
             ChangeStateTo(FireWallState.Active );
         return RedirectToAction("index");
     }
}
```

IFireWall.ChangeStateTo method

You can change the state of the firewall at runtime by calling this method in your code. Please note that you need to have a licensed version to be able to use this feature. The above code sample shows how this could be implemented.

Firewall stability

Access to Runtime Errors

You can access Runtime errors via the report or direct from the IFireWall.Exceptions property. Normally the list should contain no entries. If the list is not empty then these are the errors that will have been sent to us for fixing if you have joined the customer improvement program. Exceptions are also stored on disk on the location specified as AppDataFolder in your configuration (default is ~\App_Data\). You can always inform us of these issues by sending these to us manually.

Access to configuration errors

IFireWall.ToDo provides access to the item's that are configured incorrectly during the operation of the firewall. You can access this from the reporting interface or directly from the interface.

```
[Authorize]
public class ManagementController : Controller
{
    private IFireWall _fireWall;
    public ManagementController( IFireWall fireWall)
    {
        _fireWall = fireWall;
    }
    [CrossSite]
    public IActionResult Stability()
    {
         var model = new FireWallModel(_fireWall.Exceptions, _fireWall.ToDo)
        return View(model);
    }
}
```

You can inject IFireWall directly into the razor page and access the firewall interface directly from within in your view.

Using Firewall Plugins'

The firewall package can get additional functionality by registering the Add-Ons using the NuGet packages mentioned in this chapter. You can configure the via the extension methods on the `IFireWallServiceCollection` after having the using clause `Microsoft.Extensions.DependencyInjection` which is already included in Startup.cs

Geography

These NuGet packages enable the firewall to be geographically aware, you can configure only 1 of these packages as they all perform the same activity. Some packages only provide country information where others allow a more granular detailed dataset. You can develop your own by implementing IGeoFactory for country-level accuracy or ILatLongRepository for city-level accuracy. Have a look at the Walter.Web.FireWall.Geo namespace for information on the data types and base classes, you can access the document using this link.

> It is advisable to use MemoryCashing as it does make sense to cash this data. Have a look at the firewall configuration option for cashing GeoLocation data here.

Walter.Web.FireWall.Geo.MaxMind

Implements both IgeoFactory and ILatLongRepository depending on the MaxMind files you have downloaded or the API license you have subscribed to with MaxMind. MaxMind works with the free GeoLite2-City.mmdb and GeoLite2-Country.mmdb files that you have to license MaxMind. The only thing you need to do is point it to a shared directory or no directory at all and the firewall's Data Directory is assumed, if nothing is provided, then the default is App_Data

If a server hosts multiple FireWall instances then these instances can all share the same files and will not block file excess if the file option is used. See the source code sample on the next page that shows how it is accessing a MaxMind folder on the server's D: drive for access to the GeoLite2-City.mmdb and GeoLite2-Country.mmdb files.

The below sample shows the use of the extension method `UseGeography` with having enabled cashing. Doing so will improve response speeds as well as lower cost when using the API.

```
services.AddMemoryCache();
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
     , domainName: new Uri("https://www.test.dll", UriKind.Absolute)
     , options =>{
      // your options
      options.Cashing.GeoLocation.SlidingExpiration =
                                            TimeSpan.FromMinutes(50);
     }).UseGeography(new System.IO.DirectoryInfo("D:\\MaxMind"));
```

When using the API there is no sharing of resources as each web application will request it's own data. You can implement your own Geographical interfaces by implementing and registering them with the dependency injection framework. The base class to implement is named GeoFactory, To have the firewall capture city-data, you need your proprietary instance of GeoFactory implements the ILatLongRepositoryrequired interface.

Walter.Web.FireWall.Geo.Native

Only provides data when calling IGeoFactory.QueryLocation or IGeoFactory.TryGetLocation and is using the DNS system to resolve IP address location via the ISP provider of the IP address. If using the Native package you only state UseGeography() as it takes no parameters.

```
services.AddMemoryCache();

services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey

     , domainName: new Uri("https://www.test.dll", UriKind.Absolute)
     , options =>{

      // your options
      options.Cashing.GeoLocation.SlidingExpiration =
                                            TimeSpan.FromMinutes(50);

     }).UseGeography();
```

This simple geography class is not fancy, however, you can get a license with a data subscription and have it give more details.

When using geography you will be using the resources located in the "Walter.BOM.Geo" namespace. GeoLocation enumeration is used for both regions as well as individual countries. The static GeoLocationMapping helper class is useful when dealing with countries and regions. The enum values of GeoLocation map to the ISO country codes so you can cast (int)GeoLocation to access that, and back from int to GeoLocation exceptions to this are the regions as they have negative numbers.

Due to the large number of supported countries combining them using flags are not possible even if using non CSLCompliant data types like ulong and uint.

Reporting

These NuGet packages can be combined as they all report to a different "location" they also report at different levels of detail. If you like to implement your own reporting plugin then you can do so by implementing `IReportDestination` and add you instance to the reporting destination using the `AddDestination` method from `IFireWall.Configuration.Reporting`

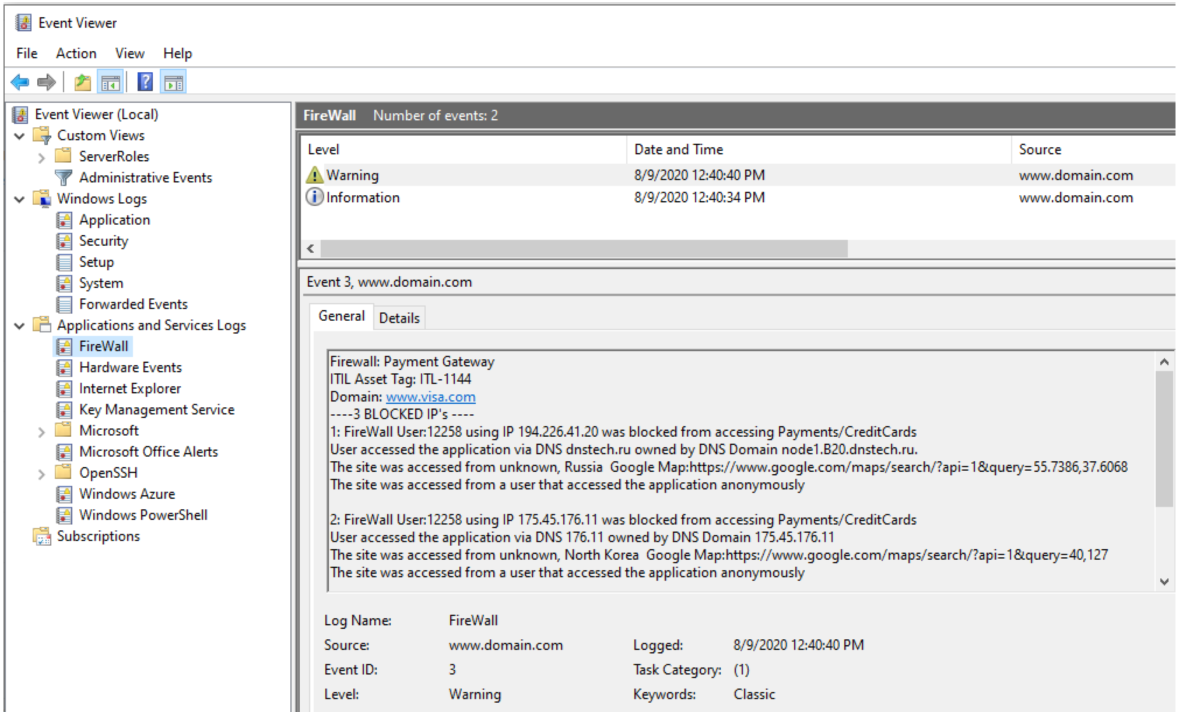
Walter.Web.FireWall.EventLog

A very simple logger that sends firewall information to the windows event log. This method is ideal when used with enterprise tools that monitor the event log like Microsoft System Center Operations Manager. See EventID class for information on Event ID and Task Category

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
    , domainName: new Uri("https://www.domain.com", UriKind.Absolute)
    , options =>
    {
      //your options
    }).UseEventLogLogging(
        options.LogName = "FireWall";
        options.SourceName = "www.domain.com";

);
```

A sample of what such an event would look like is shown in the picture on the next page. It shows an EventView source named FireWall that reports on a given domain. You can monitor your SLA's using these constants.



FireWall Number of events: 2

Level	Date and Time	Source
Warning	8/9/2020 12:40:40 PM	www.domain.com
Information	8/9/2020 12:40:34 PM	www.domain.com

Event 3, www.domain.com

```
Firewall: Payment Gateway
ITIL Asset Tag: ITL-1144
Domain: www.visa.com
----3 BLOCKED IP's ----
1: FireWall User:12258 using IP 194.226.41.20 was blocked from accessing Payments/CreditCards
User accessed the application via DNS dnstech.ru owned by DNS Domain node1.B20.dnstech.ru.
The site was accessed from unknown, Russia  Google Map:https://www.google.com/maps/search/?api=1&query=55.7386,37.6068
The site was accessed from a user that accessed the application anonymously

2: FireWall User:12258 using IP 175.45.176.11 was blocked from accessing Payments/CreditCards
User accessed the application via DNS 176.11 owned by DNS Domain 175.45.176.11
The site was accessed from unknown, North Korea  Google Map:https://www.google.com/maps/search/?api=1&query=40,127
The site was accessed from a user that accessed the application anonymously
```

Log Name:	FireWall		
Source:	www.domain.com	Logged:	8/9/2020 12:40:40 PM
Event ID:	3	Task Category:	(1)
Level:	Warning	Keywords:	Classic

Walter.Web.FireWall.SMTPLogger

The SMTP logger is packed with quite a few functions, one of them naturally includes sending email however there are quite a few more compliance features that you may find useful.

```
services.AddFireWall(“License”,”Key”
    , domainName: new Uri("https://www.domain.com", UriKind.Absolute)
        , options =>{
        //your firewall options
        options.WebServices.CSPReportUrl = new Uri(“api/csp”);
    }).UseSMTPReportingDatabase(“DatabaseConnections”
        , options =>{
        //your SMTP options
    });
```

The SMTP logger will use a disk or database-backed buffer to stage data in and send the to be reported issues to the email address at intervals defined in the email profiles. The backing storage for the SMTP logger can be disk-based when using UseSMTPReportingDisk or make use of a database as shown in the previous code samples.

Walter.Web.FireWall.SqlLogger

The most complete logging provided by a plugin is the SqlServer database logging. This uses a flexible schema where you can alter all tables as long as they do not break a stored procedure or are located in the ETL (Extract Transform and Load) schema. The logger will only execute procedures and perform bulk inserts in the ETL tables into the database. The database can be used to collect data for several firewall instances on the network. If licensed, you can also use the Always-On capabilities of SQL Server with the firewall if a higher SLA is needed.

Please note that during initialization all add-ons that use a database as backing storage will create or update the database schema. The user in the connection string will need to have DDL rights to perform these updates.

Stored procedures are always dropped and re-created on start.-up if the NuGet package is newer than the last update.

This allows the procedures execution plans to be updated for the table data volumes. A copy of the T-SQL executed is in the ApplicationData folder with the name of the Nuget Package. These T-SQL files are reference only, they are not read when performing DDL statements. Deleting the "Sql-Updates {packageName}.json"file will cause the update to execute. You can safely delete the file to force the execute the statements several times with no side effects when you are afraid the database is not updated

The below code shows how to use SQL Server logging

```
services.AddFireWall(“License”,”Key”
    , domainName: new Uri("https://www.domain.com", UriKind.Absolute)
        , options =>{
        //your firewall options
        options.WebServices.CSPReportUrl = new Uri(“api/csp”);
    }).UseFireWallReportingDatabase(“DatabaseConnections”);
```

The database scripts for the reporting database are located in the .\bin\Setup\SqlLogger location. You can alter the database to fit your requirements, however the stored procedures will be re-created on startup. Tables will be re-created if missing. Making changes to the scripts has no effect on the DDL schema as we provide them as reference only and do not execute them.

Walter.Web.FireWall.DiskLogger

The disk logger will export events to a folder on disk in a fixed csv format. This csv format can then be processed by the included powershell scripts to make changes to your servers firewall blocking ip addresses via the servers firewall.

You also have the option to start an external process, this could be an application or batch file that will post process the CSV files generated by the Logger.

Files generated are:

{ExceptionDirectory}\{yyyyMMdd}_Errors.csv allows you to look at the errors that have been raised by the firewall and contains the information needed to solve the issued raised.The file contains the following data fields:

1 Date: The date in yyyy.MMM.dd HH:mm:ss format
2 Type : The type of exception including the namespace
3 Code: The error code
4 Message: The error message
5 memberName: The method causing or raising the exception
6 class: The class name causing or raising the exception
7 Line: The line number of the exception
8 Help: Link to the help index to solve the issue if the issue is a known issue.

{DataDirectory}\BlockIP.csv containing the ip addresses blocked by the firewall.The file contains the following data fields and no header column:

1 name: The name of the incident
2 The IP address
3 The Reason flags (seperated with | character)
4 The group name that the incident belongs to

The firewall will skip any entries where the reasons are configured to be excluded in the firewall options IFireWall.Configuration.Rules.PhysicalFileWallExcludeReasons.

{DataDirectory}\BlockUsers.csv containing the ip addresses and FireWall UserId's blocked by the firewall.The file contains the following data fields and no header column:

1. FireWall UserId in the format "User-{Number}"
2. IPAddress
3. DNS Server of IP address
4. Country name as specified in Walter.BOM.Geo.GeoLocation or blank in no Geo add-on was configured. You can always use the Walter.Web.FireWall.Geo.Native to provide this data.

{DataDirectory}\AllowIP.csv containing the IP addresses that are released as the blocking duration has expired and contains no header and the following columns:

1 IP Address

2 Reason

3 The date the blocking was triggered

4 type of IP address, the name matches to the System.Net.Sockets.AddressFamily type

> You may need to have physical access to the server to be able to fully use this Add-On as it allows for execution of external code.

You have the possibility to use the included PowerShell scripts and manipulate your firewall when working with this Add-On. The following code shows how to integrate the disk logging.

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
    , domainName: new Uri("https://www.your-domain.com", UriKind.Absolute)
    , options =>
    {
       // your options
    }).UseDiskLogging(opt=> {
            opt.DataDirectory = @"D:\Web-FireWall\FireWall";
            opt.PowerShellOption = PowerShellFilesOption.GenerateIfMissing;
            opt.CommandLine = new System.Diagnostics.ProcessStartInfo(
                                               @"D:\jobs\FireWall.bat"){UseShellExecute = true};
            });
```

You can get more one how to configure disk logging using the online documentation

Walter.Web.FireWall.ILogger

This simple package uses the existing ILogger generated by your Logging package to write incidents to the logging interface. You can specify the namespace for the output and log level when integrating the logger in the firewall. More on this online

Data Sources

The firewall can make use of several data sources to better understand reputations of those that are visiting your application as well as protect it. This section briefly goes over those that are available via subscription, or configuration.

Walter.Web.FireWall.CookieStore

When making use of the cookie store you have the ability to swap the content of the cookie and send only “empty” cookies and no longer be leaking GDPR data to the world-wide-web. This also allows you to bypass the browser limitation of data that you can store in cookies.

User-Agent guard module data

You can configure how the firewall stores the user agent recognition data by way of configuration. The data stored get’s enriched by your firewall and is populated with data relevant to interactions with the firewall’s host application. **You should not use the same database for several applications.**

Native User-Agent recognition

The firewall comes packed with a bundle of user agents that are popular with spoofing bots as it is important we recognise them first, after and above that depending on your firewall subscription the data will include active user agent recognition. You can't disable this feature as it belongs to the passive protection layer of the firewall guard engine, you can however specify where to store the data as shown in the bellow example.

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
 , domainName: new Uri("https://www.your-domain.com/",
UriKind.Absolute)
 , options =>
{
   //your firewall settings

}).UseUserAgentDBStore(
    connectionString:Configuration.GetConnectionString("UserAgents")
    schema:"dbo");
```

Walter.Web.FireWall.UserAgent.UsersStack

A way to extend the firewall's browser recognition is by using usertack.com's popular subscription service that provides browser recognition services. The firewall can make use of this subscription service via the Nuget package Walter.Web.FireWall.UserAgent.UsersStack.

To use the service to enrich the data simply integrate the nuget package, register a free or paid subscription with https://www.userstack.com and replace the native UserAgent store with the storage provided by the services extension method for Userstack.com

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
 , domainName: new Uri("https://www.your-domain.com/",
UriKind.Absolute)
 , options =>
{
    //your firewall settings
}).UseUserStackDBUserAgentStore(connectionString:
                    Configuration.GetConnectionString("UserAgents"),
                    option => {
                        option.ApiKey = "123456789";
                        option.Https = false;
});
```

More about this nuget package is available online here

Advanced protection on dedicated servers

If you have a dedicated server then you can integrate the application firewall with your physical firewall. The application firewall is able to view communications and extrapolate that someone is trying to access the server via a scripted PEN attack and generate firewall rules that will block and unblock IP addresses in the physical firewall.

You can enable physical firewall protection by using the firewall's disk logging features. The below sample shows a request for the firewall to output .csv control files that can be processed by for example a powershell script. A sample PowerShell implementation can be found in the \bin\Setup\PowerShell folder.

```
services.AddFireWall(FireWallTrial.License, FireWallTrial.DomainKey
    , domainName: new Uri("https://www.your-domain.com", UriKind.Absolute)
    , options =>;
    {
        //your options
    }).UseDiskLogging(opt=>opt.Directory = @"c:\Web-FireWall\FireWall")
       .UseEventLogDataSource();
```

The method *UseEventLogDataSource* comes from installing NuGet package Walter.Net.Windows.FireWall. More on this in the chapter covering external data providers.

Physical firewall are dumb, they can allow communication through a port or not. You can further limit the external IP addresses that are allowed to access a port, it can't however make a difference on what device is on the other side nor can it infer how many users are using an IP address, it might be a corporation with thousands of employees or a service in a datacenter.

Firewalls are blunt tools that do not understand context, the WAF framework allows you to bring context to communication on an otherwize open port. DiskLogging and physical Firewall integration reduces your possibilities to manage connectivity. Look at this online sample to see a sample configuration demonstrating how to limit the violations that are blocked by the physical firewall.

External Data Providers

The firewall can be configured to use external data sources for blocking logic as well as for enriching the data. The obvious one comes in the form of data subscription where he provides contextual data that flows in the rule engine, the other methods cone in the form of nuget packages that come with the firewall or can be provided to the firewall. This chapter briefly goes over a few of them.

Firewall data source NugetPackage Walter.Net.Windows.FireWall

When licensed the FireWall will start communicating with window firewall and monitor for rejected and allowed connection activity as well as connecting sources and makes a decision on allowing or rejecting the connection.

When using disk logging the firewall will see blocking and the effect it has and integrates the blocking in the penetration rules as a request pounding on the external firewall will not cause the web application to see the request but the blocking rule must stay enabled as the malicious activity is now managed by the external firewall.

The Walter.Net.Networking NuGet Package

This NuGet package comes included with every version of the firewall package and provides access to network related actions. When using the firewall you may notice some of the extension methods that the IPAddress gets as well as access the millisecond response when the firewall does DNS system queries. You can access the DNS system and use its power by accessing the IFireWall.DnsManager property.

The Walter.Net.FireWall.LookWhoisTalking NuGet Package

The NuGet package comes included with the firewall as it provides access to the firewall reports to analyse the communications that the server is having with the rest of the world. Depending on the license and the datasusbcription this may raise incidents if the firewall detects communications with known bad actors. This module can

1. Enrich data as is the case with the FireWall.Report() classes where you can see what processes are talking to whom.

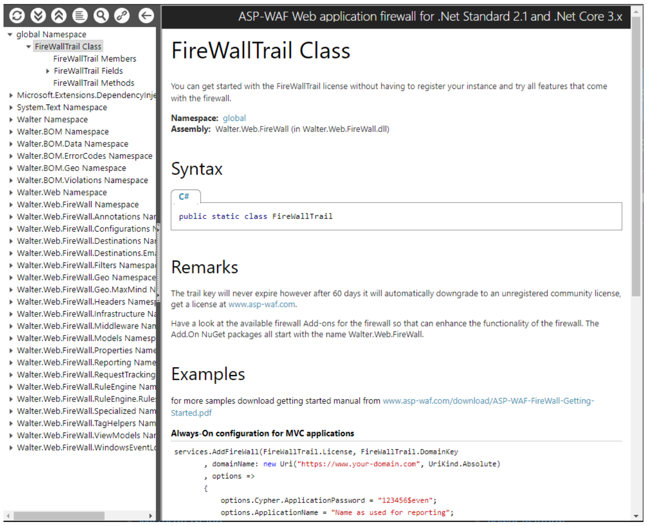
2. Raise incidents, and even terminate processes, if configured to do so on the server if trojans are detected.

Look-WhoIs-Talking also comes as a stand alone real-time platform that can be used to map out what servers are talking to other servers including the applications on either or both ends as well as what devices on your network talk to a given server. This Application mapping allows operations to create a detailed network map of application dependencies as well as give you real time data as to the user base of an application and how important servers are in your network. Look-WhoIs_talking does not need to be configured to be able to do this, it only needs a central repository to report the data to.

Documentation & Help

Getting started with a new framework is easier when there is access to good documentation. We have put a lot of effort into our documentation and provide the following resources for the firewall framework (including all add-ons). We support integration in visual studio's help system as well as online and offline help. This chapter will help you get started with each. You can access all documentation including the latest binaries when downloading ASP-WAF-FireWall.zip

Online-Help


global Namespace
FireWallTrail Class
FireWallTrail Members
FireWallTrail Fields
FireWallTrail Methods
Microsoft.Extensions.DependencyInje
System.Text Namespace
Walter Namespace
Walter.BOM Namespace
Walter.BOM.Data Namespace
Walter.BOM.ErrorCodes Namespace
Walter.BOM.Geo Namespace
Walter.BOM.Violations Namespace
Walter.Web Namespace
Walter.Web.FireWall Namespace
Walter.Web.FireWall.Annotations Nar
Walter.Web.FireWall.Configurations N
Walter.Web.FireWall.Destinations Nar
Walter.Web.FireWall.Destinations.Em
Walter.Web.FireWall.Filters Namespa
Walter.Web.FireWall.Geo Namespace
Walter.Web.FireWall.Geo.MaxMind N
Walter.Web.FireWall.Headers Namesp
Walter.Web.FireWall.Infrastructure Na
Walter.Web.FireWall.Middleware Nan
Walter.Web.FireWall.Models Namesp
Walter.Web.FireWall.Properties Name
Walter.Web.FireWall.Reporting Name
Walter.Web.FireWall.RequestTracking
Walter.Web.FireWall.RuleEngine Nam
Walter.Web.FireWall.RuleEngine.Rules
Walter.Web.FireWall.Specialized Nam
Walter.Web.FireWall.TagHelpers Nam
Walter.Web.FireWall.ViewModels Nar
Walter.Web.FireWall.WindowsEventLc

ASP-WAF Web application firewall for .Net Standard 2.1 and .Net Core 3.x

FireWallTrail Class

You can get started with the FireWallTrail license without having to register your instance and try all features that come with the firewall.

Namespace: global
Assembly: Walter.Web.FireWall (in Walter.Web.FireWall.dll)

Syntax

C#

```
public static class FireWallTrail
```

Remarks

The trail key will never expire however after 60 days it will automatically downgrade to an unregistered community license, get a license at www.asp-waf.com.

Have a look at the available firewall Add-ons for the firewall so that can enhance the functionality of the firewall. The Add.On NuGet packages all start with the name Walter.Web.FireWall.

Examples

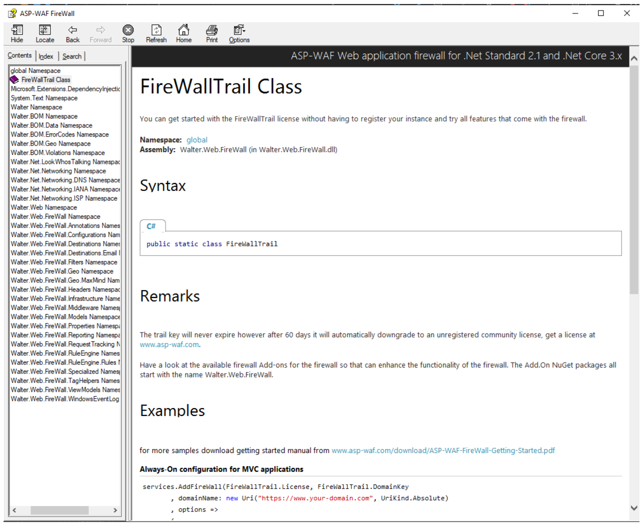
for more samples download getting started manual from www.asp-waf.com/download/ASP-WAF-FireWall-Getting-Started.pdf

Always-On configuration for MVC applications

```
services.AddFireWall(FireWallTrail.License, FireWallTrail.DomainKey
        , domainName: new Uri("https://www.your-domain.com", UriKind.Absolute)
        , options =>
        {
            options.Cypher.ApplicationPassword = "123456$even";
            options.ApplicationName = "Name as used for reporting";
```

You can access the online help for the firewall from https://firewallapi.asp-waf.com, the help files are created with GhostDock and uses an online database that makes finding a class or method easy.

Offline-Help


ASP-WAF FireWall
Hide
Locate
Back
Forward
Stop
Refresh
Home
Print
Options
Contents
Index
Search
global Namespace
FireWallTrail Class
Microsoft.Extensions.DependencyInjectic
System.Text Namespace
Walter Namespace
Walter.BOM Namespace
Walter.BOM.Data Namespace
Walter.BOM.ErrorCodes Namespace
Walter.BOM.Geo Namespace
Walter.BOM.Violations Namespace
Walter.Net.LookWhosTalking Namespac
Walter.Net.Networking Namespace
Walter.Net.Networking.DNS Namespace
Walter.Net.Networking.IANA Namespace
Walter.Net.Networking.ISP Namespace
Walter.Web Namespace
Walter.Web.FireWall Namespace
Walter.Web.FireWall.Annotations Names
Walter.Web.FireWall.Configurations Nam
Walter.Web.FireWall.Destinations Names
Walter.Web.FireWall.Destinations.Email I
Walter.Web.FireWall.Filters Namespace
Walter.Web.FireWall.Geo Namespace
Walter.Web.FireWall.Geo.MaxMind Nam
Walter.Web.FireWall.Headers Namespac
Walter.Web.FireWall.Infrastructure Name
Walter.Web.FireWall.Middleware Names
Walter.Web.FireWall.Models Namespace
Walter.Web.FireWall.Properties Namespa
Walter.Web.FireWall.Reporting Namespa
Walter.Web.FireWall.RequestTracking N
Walter.Web.FireWall.RuleEngine Names
Walter.Web.FireWall.RuleEngine.Rules I
Walter.Web.FireWall.Specialized Names
Walter.Web.FireWall.TagHelpers Names
Walter.Web.FireWall.ViewModels Names
Walter.Web.FireWall.WindowsEventLog

ASP-WAF Web application firewall for .Net Standard 2.1 and .Net Core 3.x

FireWallTrail Class

You can get started with the FireWallTrail license without having to register your instance and try all features that come with the firewall.

Namespace: global
Assembly: Walter.Web.FireWall (in Walter.Web.FireWall.dll)

Syntax

C#

```
public static class FireWallTrail
```

Remarks

The trail key will never expire however after 60 days it will automatically downgrade to an unregistered community license, get a license at www.asp-waf.com.

Have a look at the available firewall Add-ons for the firewall so that can enhance the functionality of the firewall. The Add.On NuGet packages all start with the name Walter.Web.FireWall.

Examples

for more samples download getting started manual from www.asp-waf.com/download/ASP-WAF-FireWall-Getting-Started.pdf

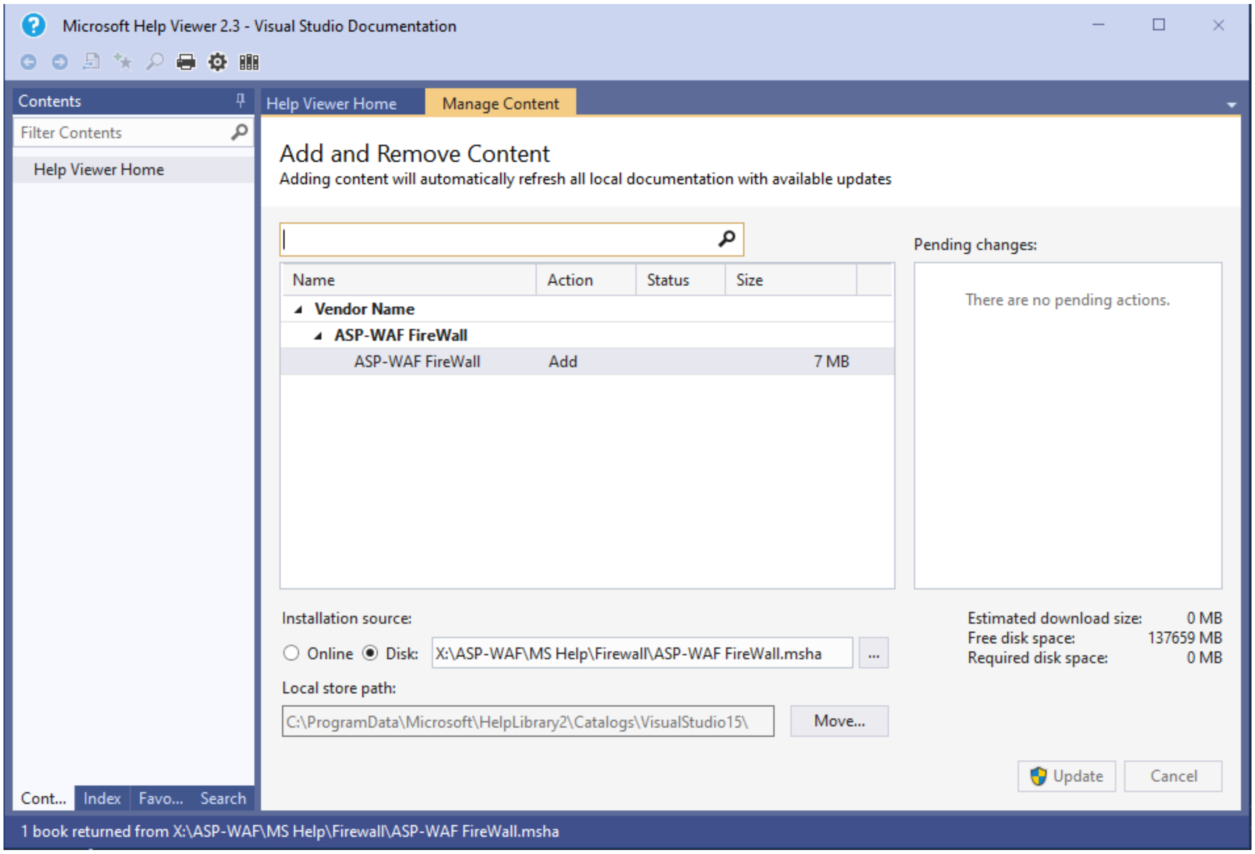
Always-On configuration for MVC applications

```
services.AddFireWall(FireWallTrail.License, FireWallTrail.DomainKey
        , domainName: new Uri("https://www.your-domain.com", UriKind.Absolute)
        , options =>
```

Offline help comes in the form of a CHM file. You can download the latest version of ASP-WAF-FireWall.chm from our website. It is quite helpful to have local help so you can access it anywhere. Important is that you store it in a location where it can execute else windows will start it with empty content. Anywhere in "my documents" or a drive other then your system drive will normally do.

Integrating with Visual studio

To integrate local help with Visual Studio open the Help → Add and Remove Help Content menu items to update the Microsoft Help Viewer. Select the “Manage Content” tab and browse to the MS Help files for the local storage path. Click “Update” to integrate the help content.



Microsoft Help Viewer 2.3 - Visual Studio Documentation

Add and Remove Content

Adding content will automatically refresh all local documentation with available updates

Name	Action	Status	Size
◢ Vendor Name			
◢ ASP-WAF FireWall			
ASP-WAF FireWall	Add		7 MB

Pending changes:

There are no pending actions.

Installation source:

○ Online ● Disk: X:\ASP-WAF\MS Help\Firewall\ASP-WAF FireWall.msha ...

Local store path:

C:\ProgramData\Microsoft\HelpLibrary2\Catalogs\VisualStudio15\ Move...

Estimated download size:	0 MB
Free disk space:	137659 MB
Required disk space:	0 MB

1 book returned from X:\ASP-WAF\MS Help\Firewall\ASP-WAF FireWall.msha

Continuing from here...

You probably have noted that there is online help as well as a downloadable version of the Firewall modules help system that can be found here. If you need more help then make use of our free installation support as well as our developers that can engage via LiveShare support and support you in your code. Send us a mail at support@asp-waf.com, or via WhatsApp on +352 661 464 601.