VB2021
localhost

7 - 8 October, 2021 / vblocalhost.com

ANATOMY OF NATIVE IIS MALWARE

Zuzana Hromcová
ESET, Canada

zuzana.hromcova@eset.com

ABSTRACT

Internet Information Services (*IIS*) [1] is *Microsoft* web server software for *Windows* with an extensible, modular architecture. It is not unknown for threat actors to misuse this extensibility to intercept or modify network traffic – the first known case of *IIS* malware targeting payment information from e-commerce sites was reported in 2013 [2].

Fast-forward to March 2021, and *IIS* backdoors are being deployed via the recent *Microsoft Exchange* pre-authentication RCE vulnerability chain (CVE-2021-26855 [3], CVE-2021-26857 [4], CVE-2021-26858 [5] and CVE-2021-27065 [6]), with government institutions among the targets. As *Outlook on the web* is implemented via *IIS*, *Exchange* email servers are particularly interesting targets for *IIS* malware.

IIS malware should be in the threat model, especially for servers with no security products. Despite this, no comprehensive guide has been published on the topic of the detection, analysis, mitigation and remediation of *IIS* malware.

In this paper, we fill that gap by systematically documenting the current landscape of *IIS* malware, focusing on native *IIS* modules (implemented as C++ libraries). Based on our analysis of 14 malware families – 10 of them newly documented – we break down the anatomy of native *IIS* malware, extract its common features and document real-world cases, supported by our full-Internet scan for compromised servers.

Rather than focusing on any single threat actor, malware family or campaign, we consider the whole class of *IIS* threats – ranging from traffic redirectors to backdoors. We cover curious schemes to boost third-party SEO by misusing compromised servers, and *IIS* proxies turning the servers into a part of C&C infrastructure.

Finally, we share practical steps for defenders to identify and remediate a successful compromise.

1. INTRODUCTION

IIS is *Microsoft* web server software for *Windows*. Since *IIS v7.0* (first shipped with *Windows Vista* and *Windows Server 2008*), the software has had a modular architecture – both *native* (C++ DLL) and *managed* (.NET assembly) modules can be used to replace or extend core *IIS* functionality [7]. For example, developers can use *IIS* modules to modify how requests are handled, or perform special logging or traffic analysis.

It comes as no surprise that the same extensibility is attractive for malicious actors – to intercept network traffic, steal sensitive data or serve malicious content. Web server software has been targeted by malware in the past (such as Darkleech [8], a malicious *Apache* module), and *IIS* software is no exception.

There have already been a few individual reports of malicious *IIS* modules, used for cybercrime and cyber espionage alike:

- 2013 – ISN infostealer reported by *Trustwave* [2], a native module.
- 2018 – RGDoor backdoor reported by *Palo Alto Networks* [9], a native module.
- 2019 – incident response report by *Secpulse* [10], native modules.
- 2020 – infostealer reported by *TeamT5* [11], a managed module.
- 2021 – IIS-Raid backdoor deployed via an *Exchange* server vulnerability, reported by *ESET* [12], a native module.

However, the existing reports on *IIS* threats are limited in scope, with the knowledge fragmented and technical details often missing or inaccurate. No comprehensive guide has been published on the topic.

In this paper, we take a step back and look at this class of threats – both known and newly reported. To limit the scope of this research, we focus on malicious *native* modules – malicious C++ libraries, installed on the *IIS* server as its modules.

We don't cover managed modules, or other malware that is able to run on *IIS* servers but that is not designed as *IIS* server modules (such as scripts). Unless explicitly stated otherwise, when the terms *IIS modules* or *modules* are used in this paper, we are always referring to *native IIS modules.*

We analyse 14 individual malware families (including 10 newly documented), obtained either from our telemetry or from *VirusTotal*.

In Section 2 of this paper, we document common *IIS* malware features, attack scenarios, prevalence and targets, based on the analysis and results of Internet scans we ran to complement our telemetry and identify additional victims.

In Section 3, we provide the essentials for reverse-engineering native *IIS* malware. We dissect the anatomy of malicious native *IIS* modules and examine how their features can be implemented. Throughout the paper we use examples taken from various malware families to illustrate the techniques and functionality and show notable cases.

Due to length limitations, full analyses of all the *IIS* malware families we have studied are provided in the extended version of this paper [13], provided as reference material.

2. BACKGROUND INFORMATION

In the course of our research, we collected 80+ unique native *IIS* malware samples and clustered them into 14 malware families. Throughout the paper, we refer to these families as Groups 1 to 14.

With the exception of the previously reported families ISN, RGDoor and IIS-Raid, the families are relatively new, with first-detected activity ranging between 2018 and 2021. Many of these families have been under active development throughout 2021, continuing as of this writing, but are not related to each other. They are individual malware families with one key feature – that they are developed as malicious native *IIS* modules.

We don't focus on attribution in this paper, and our grouping to 14 malware families doesn't necessarily correspond directly to 14 distinct threat actors. For example, while the features of Groups 8 – 12 vary, code overlaps suggest a common developer behind these families. On the other hand, several threat actors have been using an *IIS* backdoor derived from the same publicly available code, and we refer to all of these cases collectively as Group 1.

2.1 IIS malware types

Being a part of the server allows the cybercriminals to intercept traffic and bypass SSL/TLS – even if the communication channel is encrypted, the attackers have full access to data processed by the server, such as credentials and payment information processed by e-commerce sites.

Furthermore, our research shows that *IIS* modules are used to serve malicious content, manipulate search engine algorithms, or to turn benign servers into malicious proxies, which are then used in other malware campaigns to conceal C&C infrastructure.

Finally, while *IIS* is not the most widely used[1] web server software, it is used to implement *Outlook on the web* (aka *OWA*[2]) for *Microsoft Exchange* email servers, which also makes it a particularly interesting target for espionage.

We queried the *Shodan* service for servers with the *IIS* banner `X-AspNet-Version` and `Outlook` in the title to estimate a number of such servers. As shown in Figure 1, the number of public-facing servers with *OWA* running *Microsoft Exchange 2013* or *2016* is over 200,000.

Figure 1: Shodan result for public servers with OWA running Microsoft Exchange 2013 or 2016.

In all cases, the main purpose of *IIS* malware is **to process HTTP requests incoming to the compromised server and affect how the server responds to (some of) these requests** – how they are processed depends on the malware type.

We identified five main modes in which *IIS* malware operates:

- *Backdoor mode* allows the attackers to remotely control the compromised computer with *IIS* installed.
- *Infostealer mode* allows the attackers to intercept regular traffic between the compromised server and its legitimate visitors, to steal information such as login credentials and payment information.
- *Injector mode* is where *IIS* malware modifies HTTP responses sent to legitimate visitors to serve malicious content.
- *Proxy mode* turns the compromised server into an unwitting part of the C&C infrastructure for another malware family, and misuses the *IIS* server to relay communication between victims and the real C&C server.
- *SEO fraud mode* is where *IIS* malware modifies the content served to search engines to manipulate SERP algorithms and boost ranking for selected websites.

[1] According to the latest *Netcraft* web server survey [14] and *W3Techs* survey statistics [15], as of this writing, *IIS* has market share of 6-7% of websites.

[2] Previously known as *Outlook Web Access*, thus the *OWA* acronym.

These mechanisms are illustrated in Figure 2 and described in detail later in this paper.

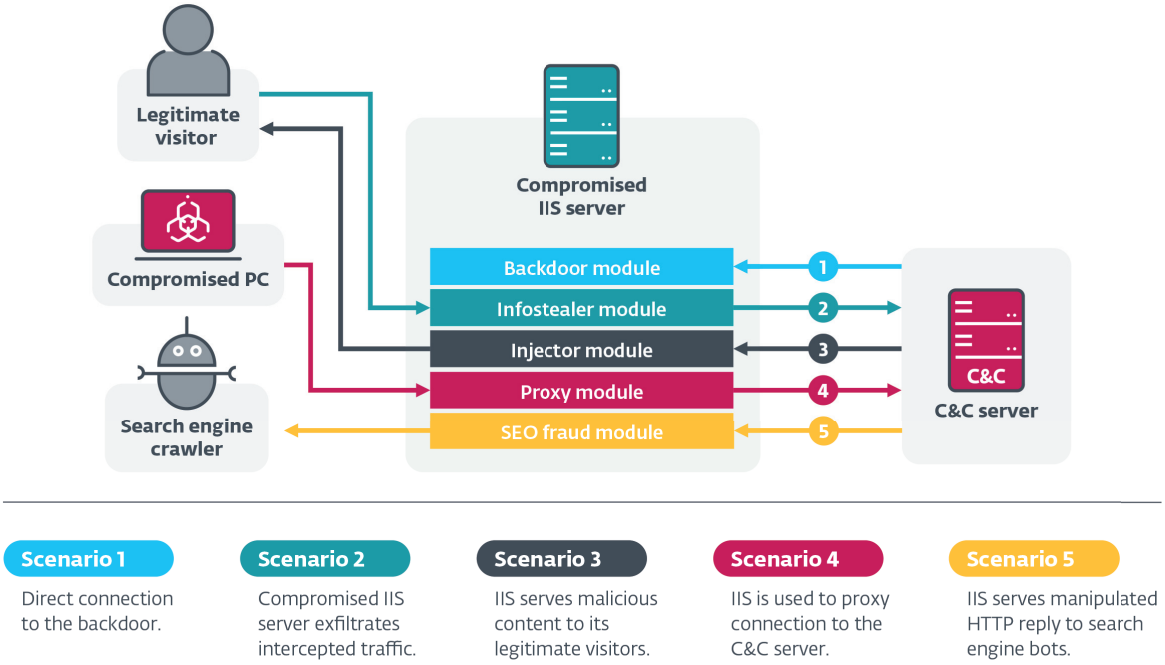

Figure 2: Overview of IIS malware mechanisms.

Of the 14 malware families examined, several combine two, three, or more of these mechanisms, as listed in Table 1. The design of *IIS* modules allows them to handle various HTTP requests differently to support several modes – for example, requests from legitimate users can be handled in infostealer mode while attacker requests are handled in backdoor mode.

Malware family	Supported modes
Group 1 (IIS-Raid)	Backdoor and infostealer
Group 2	Backdoor
Group 3	Backdoor
Group 4 (RGDoor)	Backdoor
Group 5	Infostealer
Group 6 (ISN)	Infostealer
Group 7	Backdoor
Group 8	Backdoor
Group 9	Proxy and SEO fraud
Group 10	SEO fraud
Group 11	Backdoor, proxy, SEO fraud and injector
Group 12	Backdoor, proxy, SEO fraud and injector
Group 13	Backdoor and SEO fraud
Group 14	SEO fraud and injector

Table 1: IIS malware families studied in this paper.

2.2 Typical attack overview

2.2.1 Initial vector

The raw data we had for this research was mostly malware samples only, often missing contextual information. Thus, it is difficult to determine the initial access vector used to install these malicious modules into *IIS* servers.

However, we know that administrative rights are required to install a native *IIS* module, as they have unrestricted access to any resource available to the server worker process [7]. This narrows down the options for the initial attack vector.

We have seen evidence for two scenarios:

Trojanized modules

The first observed initial access technique is through trojanized *IIS* modules. In this scenario, the *IIS* server administrator unwittingly installs a trojanized version of a legitimate *IIS* module, perhaps one downloaded from unofficial sources. For example, Group 12 includes a trojanized version of a legitimate native module called *F5 X-Forwarded-For*, that converts the `X-Forwarded-For` HTTP header so that it's visible as the client IP address in the *IIS* logs.

Server exploitation

Another option is an attacker who is able to get access to the server via some configuration weakness or vulnerability in a web application or the server, and then installs the malicious *IIS* module on the server once such access is gained.

According to our telemetry, Group 7 samples are used in connection with JuicyPotato (detected as Win64/HackTool.JuicyPotato by *ESET* security solutions), which is a privilege escalation tool. Furthermore, in March 2021 we detected several variants of Group 1 samples (based on an open-source *IIS* backdoor and used by various actors) deployed on vulnerable *Microsoft Exchange* servers via the ProxyLogon vulnerability chain (CVE-2021-26855, CVE-2021-26857, CVE-2021-26858, and CVE-2021-27065).

2.2.2 Persistence and execution

Once installed, a native *IIS* module is loaded by all worker processes on the server. *IIS* Worker Process (`w3wp.exe`) handles the requests sent to the *IIS* web server; thus an *IIS* module is able to affect the processing of every request [16].

IIS itself is persistent – with the default installation, its services (such as *World Wide Web Publishing Service*, *Windows Process Activation Service* and *Application Host Helper Services*) are configured to run automatically at each system start. This means there is no need for native *IIS* malware to implement additional persistence mechanisms.

2.3 Victimology

According to our telemetry, only a small number of servers were targeted by the studied malware families, but this is likely affected by our limited visibility into *IIS* servers – it is still common for administrators not to use any security software on servers.

To complement our telemetry, we therefore performed Internet-wide scans for selected families to identify other potential victims.

It is important to note that victims of *IIS* malware are not limited to compromised servers – all legitimate visitors to the websites hosted by these servers are potential targets, as the malware can be used to steal sensitive data from the visitors or serve malicious content.

We will not list all the targets exhaustively in this section. Instead, we will focus on the most notable case – Group 1 – which is a collection of malware samples derived from a publicly available backdoor called IIS-Raid. In its original form, the backdoor supports simple features such as downloading and uploading files, and running shell commands, which can be activated when attackers send an HTTP request including custom headers with a password. This malware has been customized by various threat actors – we have found 11 header and password combinations.

In March 2021, we detected a wave of IIS-Raid variants in the wild after the *Microsoft Exchange* pre-authentication RCE vulnerability chain (CVE-2021-26855, CVE-2021-26857, CVE-2021-26858 and CVE-2021-27065), a.k.a. ProxyLogon [17], was disclosed. Several threat actors have used this vulnerability chain to, inter alia, deploy *IIS* malware on *Exchange* servers that have *OWA* support that relies on *IIS*. We have already reported one variant[3] of IIS-Raid in our blog post about how this vulnerability is being used by various threat actors to compromise *Microsoft Exchange* servers around the world [12].

Since then, we have detected three more variants of IIS-Raid, and an additional variant of the Group 3 backdoor, all spreading through the vulnerability to *Microsoft Exchange* servers.

One of these samples was named deceptively `Microsoft.Exchange.Clients.OwaAuth.dll`, another had the following PDB path embedded:

```
C:\ProxyLogon-CVE-2021-26855-main\IIS-Raid-modify\module\x64\Release\IIS-Backdoor.pdb
```

Figure 3 shows the geographical locations of servers affected by these five campaigns, using the data from our telemetry and Internet-wide scans.

[3] SHA-256: `A11626D55EE9C958D86E8C77DFE19F66CDF545FBD8743126081F46DC24446767`

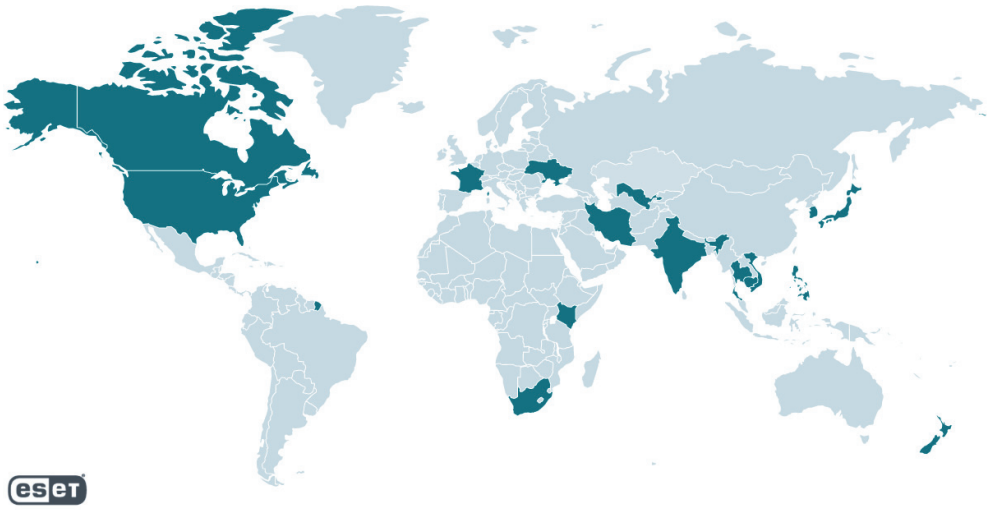

Figure 3: Victims of native IIS modules spread via the ProxyLogon vulnerability chain.

The following entities were among the victims:

- Government institutions in three countries in Southeast Asia
- A major telecommunications company in Cambodia
- A research institution in Vietnam
- Dozens of private companies in a range of industries, located mostly in Canada, Vietnam and India, and others in the USA, New Zealand, South Korea and other countries.

We notified all these victims about the compromises.

We didn't find any pattern among the targeted servers, and it is possible that these attackers used the vulnerability to compromise as many servers as possible, without any targeting. In at least one case[4], the attackers used the malware to spread the Lemon Duck cryptominer. This finding was already reported by *Sophos* [18] in May 2021, and we independently confirmed it using data from the compromised *IIS* server, which was provided to us by a victim in South Korea.

On the other hand, another version[5] derived from the same IIS-Raid source code was probably used for cyber espionage. This version has been used since at least January 2021, targeting only a small number of high-profile users in Cambodia and Vietnam. The PDB path of the analysed sample also suggests this sample was a part of a targeted campaign: `C:\Users\cambodia\Desktop\ok\ok\IIS-Raid-master\module\x64\Release\IIS-Backdoor.pdb`. In this case, the ProxyLogon vulnerability could be only one of the possible initial compromise vectors.

3. ANATOMY OF NATIVE IIS MALWARE

In this core section of our paper, we dissect the architecture of native *IIS* modules and explain how threat actors fit their malicious functionality into this architecture.

3.1 Native IIS malware essentials

A native *IIS* module is a dynamic-link library (DLL) written using the *IIS* C++ API. Native modules are located in the `%windir%\system32\inetsrv\`[6] folder on the server and can be configured for some, or for all, applications hosted by the server. These modules can be configured by a command-line tool, `AppCmd.exe`, via a GUI editor, *IIS Manager*, or by manually editing the `%windir%\system32\inetsrv\config\ApplicationHost.config` configuration file.

The modules are then loaded by the *IIS Worker Process* (`w3wp.exe`), which handles requests sent to the *IIS* server.

3.1.1 Module class

In order for *IIS* to load the DLL successfully, any native *IIS* module must export the RegisterModule [19] function, which is the library entry point, responsible for registering the module for (one or more) server *events*. Events are generated when *IIS* processes an incoming HTTP request and event *handlers* are where the core functionality of *IIS* modules is implemented.

[4] SHA-256: `A11626D55EE9C958D86E8C77DFE19F66CDF545FBD8743126081F46DC24446767`
[5] SHA-256: `17DE3F731A78BC740C5B57FB6D667CB68D93B5FE94076C852DDB30D7089988CC`
[6] Or the `%windir%\SysWOW64\inetsrv\` folder.

Therefore, all native *IIS* modules [20] (malicious or benign) will implement a module class inheriting either from `CHttpModule` [21] class or from `CGlobalModule` [22], and will override a number of their event handler methods, listed in Figure 4.

```
; const HttpModule::`vftable'
??_7HttpModule@@6B@ dd offset OnBeginRequest
                                        ; DATA XREF: sub_7454A310+19↑o
                                        ; sub_7454A360+9↑o
                dd offset OnPostBeginRequest
                dd offset OnAuthenticateRequest
                dd offset OnPostAuthenticateRequest
                dd offset OnAuthorizeRequest
                dd offset OnPostAuthorizeRequest
                dd offset OnResolveRequestCache
                dd offset OnPostResolveRequestCache
                dd offset OnMapRequestHandler
                dd offset OnPostMapRequestHandler
                dd offset OnAcquireRequestState
                dd offset OnPostAcquireRequestState
                dd offset OnPreExecuteRequestHandler
                dd offset OnPostPreExecuteRequestHandler
                dd offset OnExecuteRequestHandler
                dd offset OnPostExecuteRequestHandler
                dd offset OnReleaseRequestState
                dd offset OnPostReleaseRequestState
                dd offset OnUpdateRequestCache
                dd offset OnPostUpdateRequestCache
                dd offset OnLogRequest
                dd offset OnPostLogRequest
                dd offset OnEndRequest
                dd offset OnPostEndRequest
                dd offset OnSendResponse
                dd offset OnMapPath
                dd offset OnReadEntity
                dd offset OnCustomRequestNotification
                dd offset OnAsyncCompletion
                dd offset Dispose
                dd offset sub_7454A360
                dd offset ??_R4HttpModuleFactory@@6B@ ; const HttpModule
; const HttpModuleFactory::`vftable'
??_7HttpModuleFactory@@6B@ dd offset sub_7454A310
```

```
; const CMyGlobalModule::`vftable'
??_7CMyGlobalModule@@6B@ dq offset OnGlobalStopListening
                                        ; DATA XREF: DNameNode
                                        ; Terminate+5↑o
                dq offset OnGlobalCacheCleanup
                dq offset OnGlobalCacheOperation
                dq offset OnGlobalHealthCheck
                dq offset OnGlobalConfigurationChange
                dq offset OnGlobalFileChange
                dq offset OnGlobalPreBeginRequest
                dq offset OnGlobalApplicationStart
                dq offset OnGlobalApplicationResolveModules
                dq offset OnGlobalApplicationStop
                dq offset OnGlobalRSCAQuery
                dq offset OnGlobalTraceEvent
                dq offset OnGlobalCustomNotification
                dq offset Terminate
                dq offset OnGlobalThreadCleanup
                dq offset OnGlobalApplicationPreload
                dq offset OnSuspendProcess
```

Figure 4: Module class methods of `CHttpModule` class (left) and `CGlobalModule` class (right).

The first step of analysing a malicious native *IIS* module is to locate the module class and identify the overridden methods – this is where the malicious functionality will be implemented. The default method implementations are easy to identify, as they only print a debug message, similar to the message illustrated in Figure 5.

> 'This module subscribed to event CHttpModule::OnSendResponse but did not override the method in its CHttpModule implementation. Please check the method signature to make sure it matches the corresponding method.'

```
.text:00000000001417A0
.text:00000000001417A0
.text:00000000001417A0
.text:00000000001417A0 OnSendResponse proc near
.text:00000000001417A0 sub     rsp, 28h
.text:00000000001417A4 lea     rcx, OutputString ; "This module subscribed to event "
.text:00000000001417AB call    cs:OutputDebugStringA
.text:00000000001417B1 lea     rcx, aChttpmoduleOns ; "CHttpModule::OnSendResponse"
.text:00000000001417B8 call    cs:OutputDebugStringA
.text:00000000001417BE lea     rcx, aButDidNotOverr ; " but did not override the method in its"...
.text:00000000001417C5 call    cs:OutputDebugStringA
.text:00000000001417CB call    cs:DebugBreak
.text:00000000001417D1 xor     eax, eax
.text:00000000001417D3 add     rsp, 28h
.text:00000000001417D7 retn
.text:00000000001417D7 OnSendResponse endp
.text:00000000001417D7
```

Figure 5: Default event handler method `CHttpModule::OnSendResponse`.

In the malware families we examined, the malicious functionality is usually implemented across one to three event handlers, with the rest of the methods not overridden. For example, Group 7 implements its malicious functionality in the `OnBeginRequest`, `OnLogRequest` and `OnSendResponse` handlers; Group 5 only overrides the `OnPostBeginRequest` handler.

However, not all implemented handlers are necessarily malicious. For example, Group 12 includes a trojanized version of a legitimate native module called *F5 X-Forwarded-For*. This module converts the `XForwardedFor` HTTP header so it's visible as the client IP address in the *IIS* logs, which is implemented as `OnAcquireRequestState`, `OnSendResponse` handlers. Group 12 malware builds its code on the same module, with added malicious functionality, as the `OnBeginRequest` handler.

Both Group 7 and Group 12 handlers are shown in Figure 6.

Figure 6: Group 7 (left) and Group 12 (right) event handler methods.

3.1.2 Request-processing pipeline

As the previous section explains, the `RegisterModule` function is responsible for initialization while most of the malicious functionality in native *IIS* malware is found in its event handlers. This section explains the significance of these events and when they are triggered.

Events are steps in which *IIS* processes all incoming HTTP requests (whether GET, POST or other). These steps are taken serially, in the *request-processing pipeline* illustrated in Figure 7, and each of them generates two *request-level notifications* for the request [16], [23].

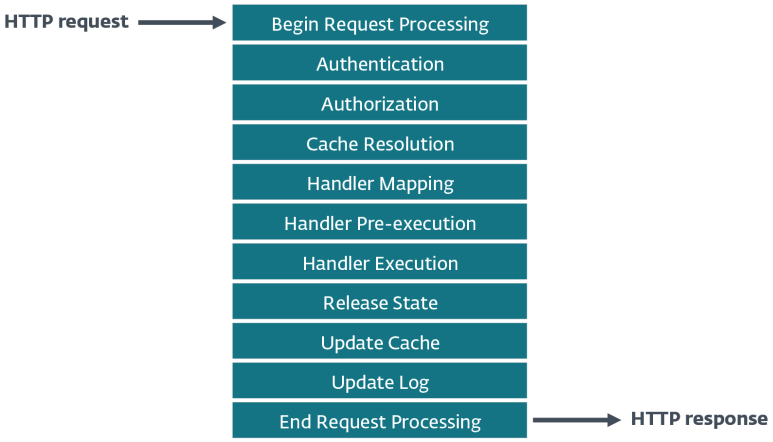

Figure 7: HTTP request-processing pipeline in IIS.

For example, the first step (`BeginRequest` event) generates:

- Event notification handled by the `OnBeginRequest` handler
- Post-event notification handled by the `OnPostBeginRequest` handler.

Post-event notifications are generated immediately after the corresponding request-level event in the pipeline. (See *Microsoft* documentation [21] for the full list of request-level notifications.) Using these notifications, a malicious *IIS* module can hook any part of the pipeline.

Other notifications are generated when specific, non-deterministic events occur. Most notably, the `OnSendResponse` handler handles the event when *IIS* sends the response buffer, which is a step with no fixed position in the pipeline.

For malicious modules, the difference between event and post-event request notifications is generally not significant (except for some corner cases). In our sample set, the malware generally registers handlers at the beginning of the pipeline (to be able to process the incoming requests), and/or when a response is being sent (to be able to intercept or modify it).

Finally, some server events are not tied to individual HTTP requests but occur on the global level. For example, the `GlobalFileChange` event occurs when a file within a website is changed. Some Group 9 samples override the `CGlobalModule::OnGlobalPreBeginRequest` method to process requests before they enter the request-processing pipeline. (See *Microsoft* documentation [22] for the full list of global-level notifications.) An *IIS* module can register for both request-level and global-level notifications, as is the case with the Group 9 malware.

Malicious modules will override a combination of these event handler methods, for example to intercept legitimate traffic or to handle incoming requests from the C&C server.

3.1.3 RegisterModule function

To summarize, each native *IIS* module must export the *RegisterModule* [19] function and implement at least one of these classes [20]:

a) To be able to register for request-level notifications:
A class inheriting from *CHttpModule* [21] (module class) and a class implementing *IHttpModuleFactory* [24] (the factory class for the module). The factory class is responsible for creating instances of the module for each incoming HTTP request.

b) To be able to register for global-level notifications:
A class inheriting from *CGlobalModule* [22].

The `RegisterModule` function creates instances of the core classes and registers for events that should be handled by the module. This is done by calls to the `SetRequestNotifications` or `SetGlobalNotifications` methods on the `pModuleInfo` instance, respectively, specifying a bitmask [25] of events to which it will receive notifications. (The bitmask will correspond directly to functions overridden by the module's main class.)

```
typedef HRESULT(WINAPI* PFN_REGISTERMODULE)(
    DWORD dwServerVersion,
    IHttpModuleRegistrationInfo* pModuleInfo,
    IHttpServer* pGlobalInfo
);
```

Listing: `RegisterModule` function prototype.

In the malware families we examined, a typical `RegisterModule` function is as minimalistic as in Figure 8 (used in Group 1, only registering the `OnSendResponse` handler).

```
.text:000007FEFB1F17D0 ; Exported entry   1. RegisterModule
.text:000007FEFB1F17D0
.text:000007FEFB1F17D0
.text:000007FEFB1F17D0 ; rdx [in] = IHttpModuleRegistrationInfo
.text:000007FEFB1F17D0
.text:000007FEFB1F17D0 public RegisterModule
.text:000007FEFB1F17D0 RegisterModule proc near
.text:000007FEFB1F17D0 push    rbx
.text:000007FEFB1F17D2 sub     rsp, 20h
.text:000007FEFB1F17D6 mov     ecx, 8          ; Size
.text:000007FEFB1F17DB mov     rbx, rdx
.text:000007FEFB1F17DE call    MyHttpModuleFactory_ctor
.text:000007FEFB1F17E3 lea     rcx, ??_7CMyHttpModuleFactory@@6B@ ; const CMyHttpModuleFactory::`vftable'
.text:000007FEFB1F17EA mov     cs:pFactory, rax
.text:000007FEFB1F17F1 xor     r9d, r9d
.text:000007FEFB1F17F4 mov     r8d, RQ_SEND_RESPONSE
.text:000007FEFB1F17FA mov     rdx, rax
.text:000007FEFB1F17FD mov     [rax], rcx
.text:000007FEFB1F1800 mov     rcx, rbx
.text:000007FEFB1F1803 mov     r10, [rbx]
.text:000007FEFB1F1806 add     rsp, 20h
.text:000007FEFB1F180A pop     rbx
.text:000007FEFB1F180B jmp     [r10+IHttpModuleRegistrationInfoVtbl.SetRequestNotifications]
.text:000007FEFB1F180B RegisterModule endp
.text:000007FEFB1F180B
```

Figure 8: A typical `RegisterModule` function of a native IIS module.

Optionally, the `RegisterModule` function can also set the request-level priority for the module using the `IHttpModuleRegistrationInfo::SetPriorityForRequestNotification` method. For cases when several *IIS* modules (malicious and regular) are registered for the same event, this priority is used to enforce the order in which their respective handlers will be called. For example, Group 7 registers its handlers with `PRIORITY_ALIAS_FIRST`, as shown in Figure 9. That means that this malicious module will be executed **before** all other modules on `BeginRequest` and `LogRequest` events, and **after** all other modules on the `SendRequest` event. This allows the malware to have the final word in what response will be sent to the HTTP request.

```
.text:7454A2D0 mov     eax, [edi]
.text:7454A2D2 mov     ecx, edi        ; pModuleInfo
.text:7454A2D4 push    offset aFirst   ; "FIRST"
.text:7454A2D9 push    RQ_BEGIN_REQUEST or RQ_LOG_REQUEST or RQ_END_REQUEST ; dwRequestNotification
.text:7454A2DE call    [eax+IHttpModuleRegistrationInfoVtbl.SetPriorityForRequestNotification]
.text:7454A2E1 mov     esi, eax
.text:7454A2E3 test    esi, esi
.text:7454A2E5 jns     short loc_7454A2F2
```

Figure 9: `RegisterModule` function example (Group 7).

Note that the `RegisterModule` function is not necessarily always as minimalistic as in Figure 8. Since this function is executed only once, it is a good place for initialization. Figure 10 illustrates the layout of Group 9's `RegisterModule` function, which decrypts strings and initializes a global map structure to be used by the handlers.

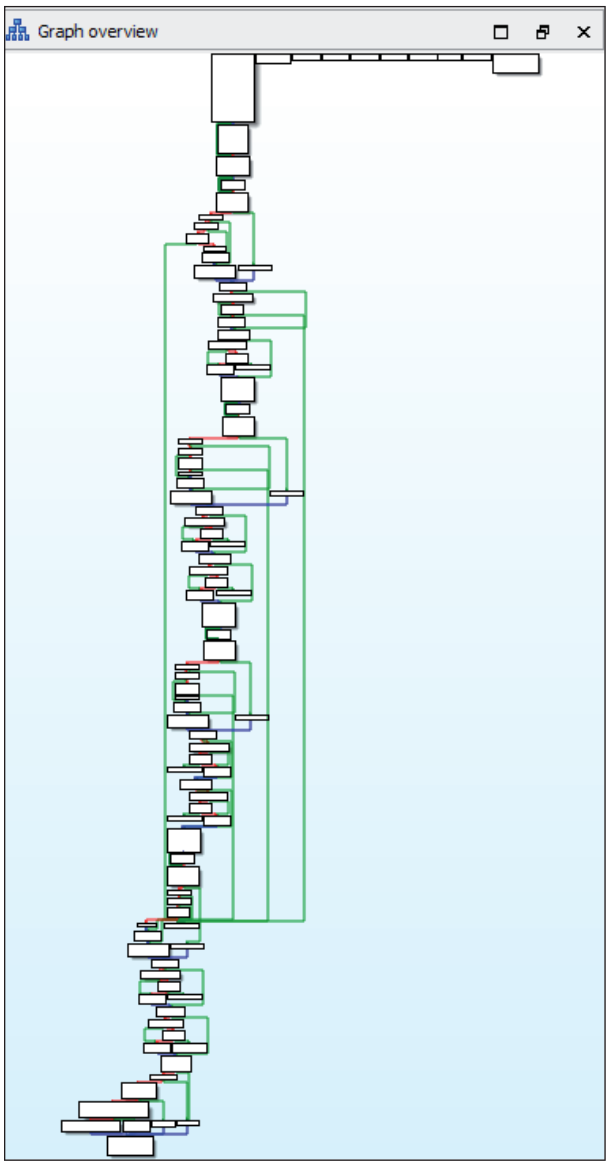

Figure 10: A more complex `RegisterModule` function with initialization functions (Group 9).

In any case, the only responsibility of the `RegisterModule` function is to initialize the module and register it for server events. To understand a malicious native *IIS* module, it is crucial to analyse the handlers.

3.2 Native IIS malware features

Whether the *IIS* malware's purpose is to steal credential information from legitimate visitors, or serve them malicious content, all native *IIS* modules operate in the same phases. As illustrated in Figure 11, a malicious *IIS* module starts with parsing an incoming HTTP request to identify whether it was sent by a legitimate user, by the attacker or another party, and

whether it should be processed. According to this classification, the malware then processes the request (for example, logs sensitive data from the request or executes a backdoor command from the request) and modifies the HTTP response accordingly. Typically, only a few of the inbound HTTP requests are of interest to the malicious module and will trigger an action, the rest of the requests pass through the malware pipeline untouched.

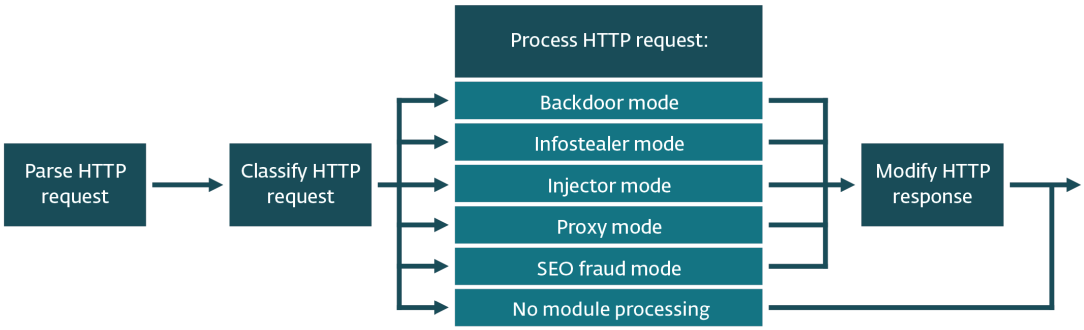

Figure 11: Typical native IIS malware phases.

In the next section, we explore how each of these phases is implemented by *IIS* handlers.

4.2.1 Parsing HTTP requests

As the first step for each incoming HTTP request, a malicious *IIS* module parses the request using instances of IHttpContext [26] and IHttpRequest [21], provided as parameters to obtain information such as requested resource, headers or request body. An example is shown in Figure 12.

```
.text:0000000180006180 readHttpBodyChunk:                    ; CODE XREF: OnBeginRequest+14F↓j
.text:0000000180006180                 mov     rdx, [rbx]
.text:0000000180006183                 mov     rcx, rbx
.text:0000000180006186                 call    [rdx+IHttpRequest2Vtbl.GetRemainingEntityBytes]
.text:000000018000618C                 test    eax, eax
.text:000000018000618E                 jz      short loc_1800061D1
.text:0000000180006190                 mov     rax, [rbx]
.text:0000000180006193                 mov     [rsp+698h+var_670], rsi
.text:0000000180006198                 lea     rcx, [rsp+698h+entityBodyBufferSize]
.text:000000018000619D                 mov     [rsp+698h+var_678], rcx
.text:00000001800061A2                 xor     r9d, r9d
.text:00000001800061A5                 mov     r8d, [rsp+698h+entityBodyBufferSize]
.text:00000001800061AA                 mov     rdx, rdi
.text:00000001800061AD                 mov     rcx, rbx
.text:00000001800061B0                 call    [rax+IHttpRequest2Vtbl.ReadEntityBody]
.text:00000001800061B6                 test    eax, eax
.text:00000001800061B8                 js      short loc_1800061D1
.text:00000001800061BA                 movsxd  r8, [rsp+698h+entityBodyBufferSize] ; Size
.text:00000001800061BF                 mov     rdx, rdi        ; void *
.text:00000001800061C2                 lea     rcx, [rsp+698h+httpRequestBody] ; Src
.text:00000001800061CA                 call    appendChunk
.text:00000001800061CF                 jmp     short readHttpBodyChunk
```

Figure 12: Reading HTTP request body (Group 2).

Another way to access HTTP headers is by using *IIS Server Variables* [27], which can be used to retrieve a specific request (client) header by using the `HTTP_<headerName>` value. This is a common way to implement HTTP request parsing in general – for example, *Apache* and *nginx* servers can both provide `HTTP_<headerName>` as environment variables. In *IIS*, the server variables can be accessed via the `GetServerVariable` method, as shown in Figure 13, where the `User-Agent`, `Referer` and `Host` headers are queried. This method of HTTP request parsing is used by Group 6, Group 11 and Group 12.

```
413  pHttpContext1 = *pHttpContext;
414  httpUserAgentValue = 0i64;
415  httpUserAgentValueLength = 0;
416  (pHttpContext1->GetServerVariable)(pHttpContext, "HTTP_USER_AGENT", &httpUserAgentValue, &httpUserAgentValueLength);
417  pHttpContext2 = *pHttpContext;
418  httpRefererValue = 0i64;
419  httpRefererValueLength = 0;
420  (pHttpContext2->GetServerVariable)(pHttpContext, "HTTP_Referer", &httpRefererValue, &httpRefererValueLength);
421  pHttpContext3 = *pHttpContext;
422  httpHostValue = 0i64;
423  httpHostValueLength = 0;
424  (pHttpContext3->GetServerVariable)(pHttpContext, "HTTP_HOST", &httpHostValue, &httpHostValueLength);
```

Figure 13: Group 11 obtains values of HTTP request headers by querying IIS server variables.

3.2.2 Classifying requests

With the information obtained from the HTTP request, the module can evaluate whether the request will be processed or ignored. That means that, as the next step, it applies its mechanisms to recognize whether the request is coming from the attacker, a legitimate user or an automated bot, as these requests will be handled differently.

Note that malicious *IIS* modules, especially *IIS* backdoors, don't usually create new connections to the C&C servers. They work in passive mode, allowing the attackers to control them by providing some 'secret' in the HTTP request. That's why these backdoors have a mechanism implemented to recognize the attacker requests, which are used to control the server and have a predefined structure.

Possibilities are broad; these are some that were used in the samples analysed:

- URL or request body matching a specific regex.
- A custom HTTP header present.
- An embedded token (in the URL, request body or one of the headers) matching a hard-coded password.
- A hash value of an embedded token matching a hard-coded value – for example, Group 12 computes the MD5 of the password in the request and compares it against a hard-coded value.
- A more complex condition – for example, a relationship between all of the above.

As an example of a more complex condition, attacker requests to control Group 7 backdoor must match the expected format, encryption and encoding, and this relationship between the URL, `Host` and `Cookie` headers:

- The malware first computes the MD5 of both the URL and `Host` header, and splits each into four double words:
  - `<h0><h1><h2><h3> = md5(Host Header value)`
  - `<r0><r1><r2><r3> = md5(Raw URL value)`
- Then, it verifies that the `Cookie` header contains a substring built from these values:
  - `<r1><h2>=<h3><r2><r3><r0><h0><h1>`
- Figures 14 and 15 illustrate a part of how this substring is assembled.

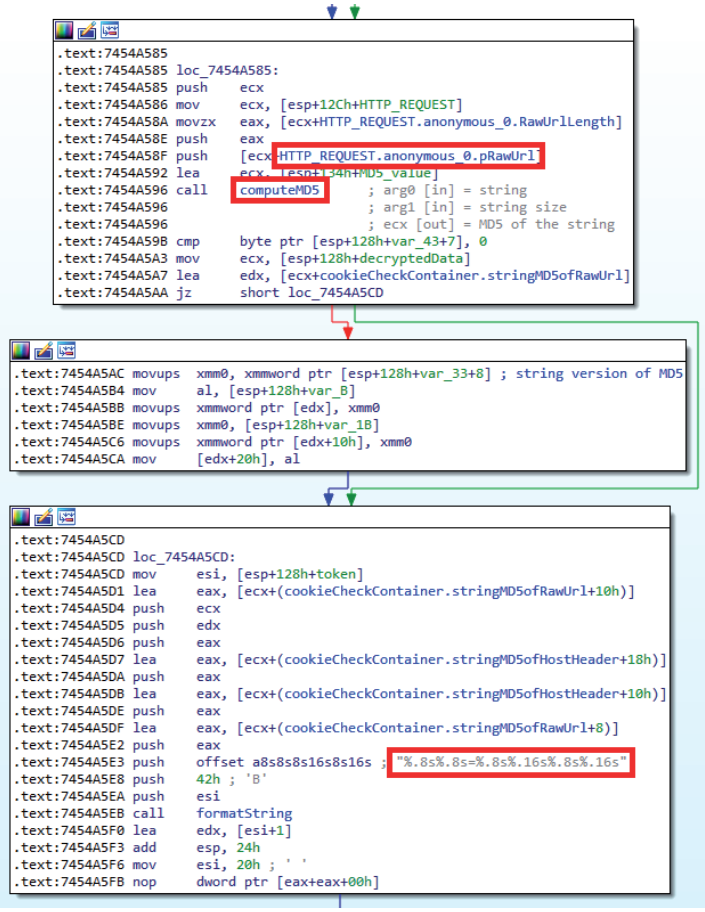

Figure 14: Attacker requests for Group 7 have a specific relationship between the request URL, `Host` and `Cookie` headers.

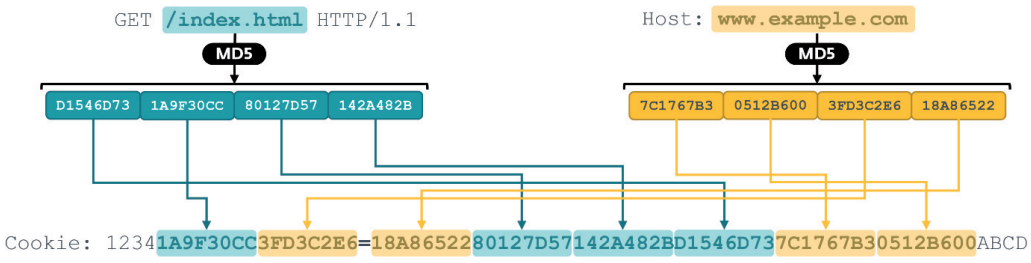

Figure 15: Group 7 backdoor attacker request format.

Other malware families are designed to manipulate search engine algorithms, and therefore need a mechanism to recognize requests from search engine crawlers. These are usually recognized by checking for specific key words in the `User-Agent` header, such as:

- 360Spider
- Baiduspider
- Bingbot
- Googlebot
- Sogou Pic Spider
- Sogou web spider
- Yahoo
- YandexBot
- YisouSpider…

3.2.3 Processing HTTP requests

In the next step, the malicious *IIS* module processes the HTTP requests – various mechanisms are used based on the malware type, as described in this section.

Infostealer mode

Infostealing *IIS* malware intercepts regular traffic between the compromised server and its clients, and exfiltrates information of interest to the attackers.

One example is Group 1, which targets HTTP requests with the string `password` in the request body, to obtain credentials from legitimate visitors. Note that, being a part of the server, these *IIS* infostealers have access to all data, even if SSL/TLS is used and the communication channel is encrypted.

Another example is malware targeting credit card information sent to e-commerce websites that don't use a payment gateway [28].

As illustrated in Figure 16, Group 5 targets HTTP POST requests made to specific URI paths (`/checkout/checkout.aspx`, `/checkout/Payment.aspx`). When a legitimate website visitor makes such a request (1), the malware logs it into a log file (2), without interfering with the HTTP reply in any way (3). At a later point, an attacker can make an HTTP request to a specific URI path (e.g. `/privacy.aspx`), with an attacker password included in the `X-IIS-Data` header (4), to exfiltrate the collected data (5, 6).

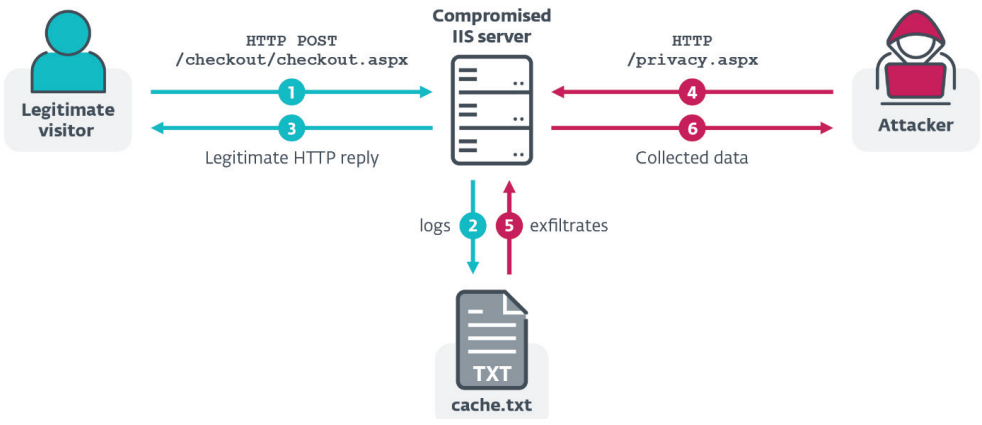

Figure 16: Group 5 infostealing mechanism.

Backdoor mode

Another class of *IIS* malware is *IIS* backdoors that allow the attacker to remotely control the compromised server by sending special HTTP requests with commands. Legitimate HTTP requests are usually ignored by the *IIS* backdoors – of course, they are handled by other (legitimate) *IIS* modules, as expected.

Note that *IIS* backdoors are implemented as *passive backdoors* – the backdoor doesn't actively request commands from the C&C server, it's the attacker who sends HTTP requests with the commands to the compromised *IIS* server.

The backdoor commands and arguments are passed as part of the URL, in the HTTP request body, or are embedded in HTTP request headers. As an example, for Group 7, the backdoor commands and arguments are embedded in the HTTP request body as key-value pairs, as shown in Table 2. The body is AES-CBC encrypted and Base64-encoded with these parameters:

- Encryption key: `DA1F8BE19D9122F6499D72B90299CAB080E9D599C57E802CD667BF53CCC9EAB2`
- IV: `668EDC2D7ED614BF8F69FF614957EF83EE`

Key	Value
`/mode`	Command type
`/action`	Command

`/path`
`/binary`
`/data`
…
	Command arguments (see Table 3 for full list)

`/credential/username`	Local user username, used for impersonation
`/credential/password`	Local user password, used for impersonation

Table 2: Group 7 attacker HTTP request body structure.

If the credentials are present, Group 7 backdoors use them to log in as that user (via the `LogonUserW`, `ImpersonateLoggedOnUser` API functions) to execute the backdoor commands on their behalf.

The command arguments are organized as nested key-value pairs, listed in Table 3. As can be seen from the table, few of the backdoor features are specific to *IIS* – it mostly supports standard backdoor commands such as:

- Get system information
- Upload/download files
- Execute files or shell commands
- Create reverse shell
- Create/list/move/rename/delete files and folders
- Map local drives to remote drives
- Exfiltrate collected data

This also applies to the other analysed *IIS* backdoor families.

Proxy mode

IIS proxies turn the compromised *IIS* servers into malicious proxies, and program them to forward requests from compromised hosts. These are a special type of malicious *IIS* modules, in that these modules are usually a part of a bigger infrastructure. Attackers may deploy this malware on compromised servers in order to build a multi-layer C&C infrastructure [29], or the *IIS* server can act like an internal proxy [30] between the C&C server and compromised machines located in a local corporate network.

As shown in Figure 17, the malicious *IIS* module recognizes a request from the compromised host (1), relays it to the C&C server (2), and then relays the obtained commands to the compromised host (3 – 4). Of course, the compromised *IIS* server doesn't necessarily have to communicate directly with the real C&C server – there can be other intermediaries to make the tracing even more difficult.

Command type (/mode value)	Command (/action value)	Arguments (key names)	Command description	Returned data (map structure or description)
init	N/A	N/A	Collects basic system information: computer name and domain, username and domain, logical drives information.	/computer/domain /computer/name /user/domain /user/name /- /name /type
file	list	/path	Collects information about the files in the specified folder.	/- /name /attr /size /create /access /write
	get	/path /binary	Downloads the file with the specified name from the compromised IIS server.	The contents of the file, encrypted and embedded within a fake PNG image (a PNG header followed by non-image data).
	create	/path /directory /data	Creates a new file or directory in the specified path. Optional /data argument can hold the file content.	/- /file /attr /size /create /access /write
	upload	/path /data	Uploads a file with the specified name to the compromised server. The /data entry contains Base64-encoded file content.	/- /file /attr /size /create /access /write
	delete	/path /files /name /attr	Deletes the list of files/directories in the given path.	/files /code /name
	move	/path /dest /copy /files /name /new	Copies or renames files from the list, from the source directory to the destination directory.	/files /code /name
	time	/path /create /access /write	Modifies file timestamps.	N/A
drive	map	/letter /share /username /password	Creates a mapping between a local and a remote drive, using the given credentials for the network resource.	N/A
	remove	/letter	Removes existing drive mapping.	N/A
cmd	exec	/cmd	Executes the specified command, either under the context of the current user, or the user provided in arguments. Returns the command output.	/output

Table 3: Group 7 backdoor commands.

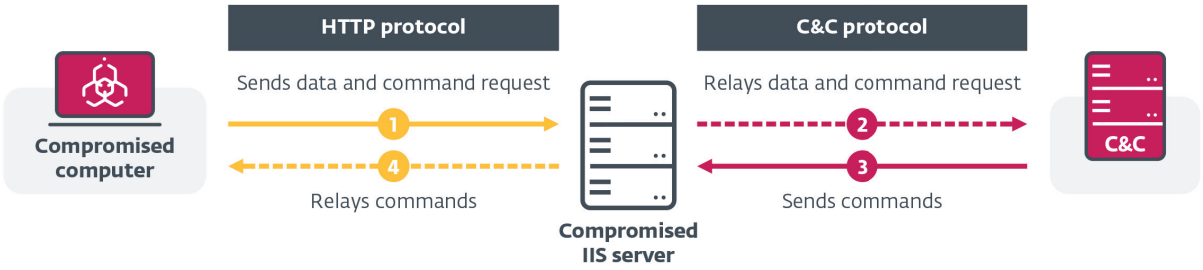

Figure 17: C&C communication leveraging a compromised IIS server as a proxy.

For example, one[7] of the Group 9 samples implements the proxy functionality in this way:

1. A compromised client makes an HTTP GET request to the compromised *IIS* server, with URI path matching this regular expression:

   ```
   ^/(app|hot|alp|svf|fkj|mry|poc|doc|20)
   ```

2. The malicious *IIS* module on the server sends an HTTP GET request to the C&C server, in this format:

   ```
   http://qp.008php[.]com/<path>|<hostHeader>|<token>|
   ```

   where `<path>` and `<hostHeader>` are the URI path and `Host` header from the original request, and `<token>` is the matched string from the regular expression (e.g. `alp`). The client IP address from the original request is also included in the headers, and some other headers are copied from the original request (`Accept-Encoding`, `Referer`).

3. The C&C server processes the request and sends a response (e.g. backdoor commands) to the compromised *IIS* server.

4. The malicious *IIS* module clears any HTTP response that could have been set by other *IIS* modules in the request-processing pipeline and replaces it with the data from the C&C server. Thus, it relays the response from the C&C server (e.g. backdoor commands) to the compromised client, without revealing the C&C server's IP address.

Note that only specific requests (coming from compromised clients) are relayed to the C&C server; requests from legitimate clients of the *IIS* server are handled normally by the server.

This technique has several advantages for the attackers:

- Using a compromised server for C&C communication can bypass detection by firewalls and some other mechanisms, as the compromised server may have a good reputation.
- Even if malware analysts or security products extract IoCs from the malicious sample, they won't point to the real C&C server.

On the negative side, since the attackers don't own the server, it may become a single point of failure and prevent victims from reaching the real C&C server if the compromised one is cleaned.

Several sophisticated threat actors have used internal or external proxies in the past (for example, GreyEnergy [31] and Duqu2 [32]), and *IIS* proxies are now another example of how this technique can be implemented.

SEO fraud mode

Another category of *IIS* malware that we analysed is pure crimeware – malware used to manipulate the content served to some visitors of the compromised *IIS* server, or to deceive search engine crawlers.

Groups 9–14 modify content served to search engine crawlers in order to artificially boost SEO for selected websites (we refer to these techniques collectively as *unethical SEO* or *SEO fraud*, although you may know the common term 'black hat SEO' [33]). The most versatile family, Group 13, supports these modes:

- Redirecting the search engines to the particular website chosen by the attacker, effectively making the compromised website a doorway page [34].
- Injecting a list of backlinks (pre-configured or obtained from the C&C server on the fly) into the HTTP response, to artificially boost its relevance/popularity (also parasitizing on the compromised website's ranking).

Note that legitimate visitors to the compromised server will still be served the expected content, so the users and the webmaster may fail to notice that something is wrong with the server.

[7] SHA-256: `A62734619EC889E7C80BB2EDC3497CD4139EBD5646DB30C20E0928B264B53435`

Group 10 uses another technique from this category:

- When a search engine web crawler is detected, this *IIS* module serves an HTML response with meta keywords and meta description tags stuffed with keywords referring to a particular *WeChat* racing group and a JavaScript script.
- The keywords are shown in Figure 18 and correspond to Chinese Unicode strings, e.g. 北京赛车微信群, 北京微信赛车群, 北京赛车微信群, PK10群, 北京8d5b车pk10微信群, which loosely translates to 'Beijing Racing WeChat Group, Beijing WeChat Racing Group, Beijing Racing WeChat Group, PK10 Group, Beijing 8d5b Car Pk10 WeChat Group'.
- The purpose of the keyword stuffing [35] could be to make the particular racing group appear more popular, and rank higher when searching for a group of this type.

```
.rdata:1002FFB0 a24494204493267 db '&#24494;&#20449;&#32676;&#45;&#36187;&#36710;&#80;&#75;&#49;&#48;'
.rdata:1002FFB0                                         ; DATA XREF: sub_10001EB0+4A↑o
.rdata:1002FFB0                                         ; .rdata:10030970↓o
.rdata:1002FFB0                 db '&#32676;&#12304;&#36827;&#32676;&#24494;&#20449;&#102;&#117;&#110'
.rdata:1002FFB0                 db ';&#53;&#55;&#54;&#52;&#52;&#12305;&#95;&#24184;&#36816;&#39134;&#'
.rdata:1002FFB0                 db '33351;&#95;&#24184;&#36816;&#50;&#56;&#32676;',0
.rdata:100300A1                 align 8
.rdata:100300A8 a21271201403618_0 db '&#21271;&#20140;&#36187;&#36710;&#24494;&#20449;&#32676;&#44;&#21'
.rdata:100300A8                                         ; DATA XREF: sub_10001EB0+45↑o
.rdata:100300A8                                         ; .rdata:10030974↓o
.rdata:100300A8                 db '271;&#20140;&#24494;&#20449;&#36187;&#36710;&#32676;&#44;&#21271;'
.rdata:100300A8                 db '&#20140;&#36187;&#36710;&#24494;&#20449;&#32676;&#44;&#80;&#75;&#'
.rdata:100300A8                 db '49;&#48;&#32676;&#44;&#21271;&#20140;&#36187;&#36710;&#112;&#107;'
.rdata:100300A8                 db '&#49;&#48;&#24494;&#20449;&#32676;&#44;&#80;&#75;&#49;&#48;&#2449'
.rdata:100300A8                 db '4;&#20449;&#32676;&#44;&#36187;&#36710;&#24494;&#20449;&#32676;&#'
.rdata:100300A8                 db '44;&#21271;&#20140;&#36187;&#36710;&#32676;&#44;',0
.rdata:1003025F                 align 10h
.rdata:10030260 a21271201403618 db '&#21271;&#20140;&#36187;&#36710;&#24494;&#20449;&#32676;&#44;&#21'
.rdata:10030260                                         ; DATA XREF: sub_10001EB0+40↑o
.rdata:10030260                                         ; .rdata:10030978↓o
.rdata:10030260                 db '271;&#20140;&#24494;&#20449;&#36187;&#36710;&#32676;&#12304;&#368'
.rdata:10030260                 db '27;&#32676;&#24494;&#20449;&#21495;&#102;&#117;&#110;&#53;&#55;&#'
.rdata:10030260                 db '54;&#52;&#52;&#12305;&#21271;&#20140;&#24494;&#20449;&#36187;&#36'
.rdata:10030260                 db '710;&#32676;&#44;&#21271;&#20140;&#24494;&#20449;&#36187;&#36710;'
.rdata:10030260                 db '&#32676;&#44;&#20960;&#30334;&#20154;&#20449;&#35465;&#22823;&#32'
.rdata:10030260                 db '676;&#44;&#36180;&#29575;&#39640;&#44;&#19979;&#20998;&#24555;&#4'
.rdata:10030260                 db '4;&#31119;&#21033;&#22810;&#44;&#27599;&#22825;&#37117;&#26377;&#'
.rdata:10030260                 db '24320;&#26426;&#31119;&#21033;&#9619;&#9619;&#9619;&#9619;&#9619;'
.rdata:10030260                 db '&#9619;&#9619;&#9619;&#9619;&#9619;&#9619;&#9619;&#9619;&#9619;&#'
.rdata:10030260                 db '9619;&#9619;&#27426;&#36814;&#22823;&#23478;&#21152;&#20837;&#212'
.rdata:10030260                 db '71;&#20140;&#24494;&#20449;&#36187;&#36710;&#32676;&#30456;&#2011'
.rdata:10030260                 db '4;&#20132;&#27969;&#12290;',0
.rdata:10030587                 align 4
.rdata:10030588 aHttpsJsBreakav db 'https://js.breakavs.com/93/jc.js',0
.rdata:10030588                                         ; DATA XREF: sub_10001EB0+3B↑o
.rdata:10030588                                         ; .rdata:1003096C↓o
.rdata:100305A9                 align 10h
.rdata:100305B0 aTitleSTitleMet db '<title>%s</title>',0Ah
.rdata:100305B0                                         ; DATA XREF: sub_10001EB0+61↑o
.rdata:100305B0                 db '<meta name = "keywords" content = "%s" />',0Ah
.rdata:100305B0                 db '<meta name = "description" content = "%s" />',0Ah
.rdata:100305B0                 db '<script src="%s"></script>',0
```

Figure 18: Strings used to deceive search engine crawlers (Group 10).

Note that it is possible that some webmasters willingly implement similar *IIS* modules with unethical SEO techniques to boost *their own* SEO statistics. Even if that were the case, and the users were aware of installing these modules, it is usually against the guidelines of search engines to serve their crawlers a different version of the website than is served to the users [36], [37].

However, the samples discussed in this report are *not* this case, as they all implement a communication channel with a C&C server to obtain configuration data (i.e. which website should be linked, etc.); and most of them also implement other malicious functionality (backdoor, proxy). We believe the attackers misuse the compromised *IIS* servers for this scheme, which they offer as a service to third parties.

Injector mode

As a final *IIS* malware type, *IIS* injectors are used to manipulate the content served to the legitimate visitors of the websites hosted by the compromised *IIS* server. For example, Group 12 replaces content displayed to visitors coming from search engines (based on keywords in the `Referer` header) and visitors browsing specific URIs with data obtained from the C&C server. This could include malicious scripts, advertisements or malicious redirects. Part of the configuration of this malware is shown in Figure 19.

```
aIfengIvcSogouS db 'ifeng|ivc|sogou|so.com|baidu|google|youdao|yahoo|bing|118114|biso'
                                        ; DATA XREF: CHttpModule_OnBeginRequest+293↑o
                                        ; CHttpModule_OnBeginRequest+110B↑o
                db '|gougou|sooule|360|sm|uc',0
                align 10h
aSogouSoComBaid db 'sogou|so.com|baidu|google|youdao|yahoo|bing|gougou|sooule|360|sm.'
                db 'cn|uc',0
                align 8
aHotcssHotjs    db 'Hotcss/|Hotjs/',0   ; DATA XREF: CHttpModule_OnBeginRequest+47D↑o
                                        ; CHttpModule_OnBeginRequest:loc_180004208↑o
                align 8
aHotimgHotpic   db 'HotImg/|HotPic/',0  ; DATA XREF: CHttpModule_OnBeginRequest:loc_1800042CF↑o
                                        ; CHttpModule_OnBeginRequest:loc_1800042FA↑o ...
                dq offset aOoooo        ; "                 _ooOoo_                 "...
                dq offset aO8888888o    ; "                o8888888o                "...
                dq offset a8888         ; "                88* . *88                "...
                dq offset asc_1800211D8 ; "                (| -_- |)                "...
                dq offset aOO           ; "                O\\ = //O                "...
                dq offset asc_180021238 ; "             ____//`---'\\____            "...
                dq offset asc_180021268 ; "           .   ' \\| |// `.              "...
                dq offset asc_180021298 ; "           / \\||| : |||// \\              "...
                dq offset asc_1800212C8 ; "         / _||||| -:- |||||- \\            "...
                dq offset asc_1800212F8 ; "           | | \\  -  /// | |              "...
                dq offset asc_180021328 ; "         | \\_| ''\\---//'' | |            "...
                dq offset asc_180021358 ; "          \\ .-\\__ `-` ___/-. /           "...
                dq offset asc_180021388 ; "       ___`. .' //--.--\\ `. . __         "...
                dq offset asc_1800213B8 ; "    ."" '< `.___\\_<|>_//___.' >'. ""      "...
                dq offset asc_1800213E8 ; "    | | : `- \\`.;`\\ _ //`;.`// - ` : |    "...
                dq offset asc_180021418 ; "      \\ \\ `-. \\_\t__\\ //__ _// .-` /      "...
                dq offset asc_180021448 ; "======`-.____`-.___\\_____//___.-`____.-`====="...
                dq offset asc_180021478 ; "                   `=---='                   "...
                dq offset asc_1800214A8 ; ".........................................."...
                dq offset unk_1800214D8
                align 20h
```

Figure 19: Group 12 processes HTTP requests based on keywords in URIs or `Referer` headers.

The mechanism behind *IIS* injectors is usually implemented in the same way as the one behind SEO fraud *IIS* malware – the malicious *IIS* module recognizes HTTP requests of interest, and injects data obtained from the C&C server into the HTTP response. The reason we consider it a separate malware type is in the affected parties:

- For SEO fraud malware, the modified content is served to search engine bots and it's the SERP algorithms that are manipulated.
- *IIS* injectors serve malicious content to the legitimate visitors, and could be used for displaying ads, mass-spreading of any malware, but also for watering hole attacks [38] targeting specific groups of users.

3.2.4 Modifying HTTP responses

In the previous sections, we described how various types of malicious *IIS* modules parse, classify and process inbound HTTP requests of various types. In this section, we cover the final phase – how *IIS* modules can modify or replace the HTTP response for these requests.

Malicious *IIS* modules can manipulate the HTTP response (as prepared by other *IIS* modules) using the `IHttpContext` and `IHttpResponse` interfaces. For example, after they handle attacker requests, Group 8 backdoors discard any HTTP response prepared by other *IIS* modules and replace it with their own. As shown in Figure 20, useful methods are `IHttpContext::GetResponse`, `IHttpResponse::Clear`, `IHttpResponse::WriteEntityChunks` and others.

```
1  int __cdecl sendResponse(const char *a1, int cHttpContext)
2  {
3    IHttpResponse **cHttpResponse1; // eax
4    IHttpResponse **cHttpResponse2; // esi
5    IHttpResponse *cHttpResponse3; // edx
6    int hResult1; // eax
7    int hResult2; // edi
8    HTTP_DATA_CHUNK httpDataChunks; // [esp+4h] [ebp-40h] BYREF
9
10   cHttpResponse1 = (*(*cHttpContext + offsetof(IHttpContext2Vtbl, GetResponse)))(cHttpContext);
11   cHttpResponse2 = cHttpResponse1;
12   if ( !cHttpResponse1 )
13     return 1;
14   cHttpResponse3 = *cHttpResponse1;
15   httpDataChunks.DataChunkType = HttpDataChunkFromMemory;
16   (cHttpResponse3->Clear)(cHttpResponse1);
17   ((*cHttpResponse2)->SetHeader)(cHttpResponse2, HttpHeaderContentType, "text/plain", 10, 1);
18   httpDataChunks.FromFileHandle.ByteRange.StartingOffset.LowPart = a1;
19   httpDataChunks.FromMemory.BufferLength = strlen(a1);
20   hResult1 = ((*cHttpResponse2)->WriteEntityChunks)(cHttpResponse2, &httpDataChunks, 1, 0, 1, &cHttpContext, 0);
21   hResult2 = hResult1;
22   if ( hResult1 < 0 )
23     ((*cHttpResponse2)->SetStatus)(cHttpResponse2, 500, "Server Error", 0, hResult1, 0, 0);
24   return hResult2;
25 }
```

Figure 20: Replacing HTTP response with own data (Group 8).

Another interesting case is Group 9 – in response to search engine web crawler requests, this malware *appends* data obtained from the C&C server to the HTTP response prepared by the *IIS* server.

But first, before any other modules start processing the request (in its `OnBeginRequest` handler set to the highest priority), this malware removes an `Accept-Encoding` header, if present, from the original web crawler HTTP request, to prevent other *IIS* modules from using compression. This step is illustrated in Figure 21.

This way, the *IIS* server will not compress the response for this request and the malware can easily inject additional data into the response without having to deal with the compression algorithm.

```
.text:743ECA8B mov     eax, [esi]
.text:743ECA8D mov     ecx, esi
.text:743ECA8F mov     eax, [eax+IHttpContext2Vtbl.GetRequest]
.text:743ECA92 call    eax
.text:743ECA94 push    HttpHeaderAcceptEncoding
.text:743ECA96 mov     ecx, eax
.text:743ECA98 mov     edx, [eax]
.text:743ECA9A call    [edx+IHttpRequest2Vtbl.DeleteHeader]
.text:743ECA9D mov     [ebp+var_132], 1
```

Figure 21: Group 9 deletes the `Accept-Encoding` header from the request to prevent other modules from using compression in the HTTP response.

3.3 Anti-analysis and detection evasion techniques

None of the samples we analysed used any complex method of obfuscation or other methods to avoid detection – we suspect the threat actors didn't implement these mechanisms because *IIS* servers often lack security solutions anyway, and because *IIS* malware is not that common or commonly analysed.

However, it is important to note that even without additional obfuscations implemented, some features of native *IIS* malware make analysis and detection harder implicitly:

- *IIS* API is based on C++ classes, rather than on *Windows* API functions, which can thwart some simple detection methods.
- The default C&C mechanism is 'passive': the attacker sends a specific HTTP request to the compromised *IIS* server (for example, with backdoor commands), and the malicious *IIS* module embeds the response in the HTTP response to this request. This makes it difficult to identify C&C servers without logs from the compromised server, as no C&C server is hard coded in those samples.

A couple of notable evasion techniques used by the analysed *IIS* malware families follow; a summary of the techniques used can be found in Table 4 (Section 3.4).

3.3.1 Obfuscation techniques

Some samples implement simple measures such string stacking, string encryption, UPX packing, or mimicking legitimate *IIS* modules. For example, Group 12 mimics a legitimate `F5XFFHttpModule.dll` module, while, as shown in Figure 22, one Group 5 samples uses a forged VERSIONINFO resource to mimic a legitimate *Windows IIS* module called `dirlist.dll`.

dir.dll (malicious)

```
1 VERSIONINFO
FILEVERSION 10,0,17763,1
PRODUCTVERSION 10,0,17763,1
FILEOS 0x40004
FILETYPE 0x2
{
BLOCK "StringFileInfo"
{
        BLOCK "000004B0"
        {
                VALUE "CompanyName", "Microsoft Corporation"
                VALUE "FileDescription", "Directory Listing handler"
                VALUE "FileVersion", "10.0.17763.1 (WinBuild.160101.0800)"
                VALUE "InternalName", "dirlist.dll"
                VALUE "LegalCopyright", "© Microsoft Corporation. All rights reserved."
                VALUE "OriginalFilename", "dirlist.dll"
                VALUE "ProductName", "Internet Information Services"
                VALUE "ProductVersion", "10.0.17763.1"
        }
}

BLOCK "VarFileInfo"
{
        VALUE "Translation", 0x0000 0x04B0
}
}
```

dirlist.dll (benign)

```
1 VERSIONINFO
FILEVERSION 10,0,18362,1
PRODUCTVERSION 10,0,18362,1
FILEOS 0x40004
FILETYPE 0x2
{
BLOCK "StringFileInfo"
{
        BLOCK "000004B0"
        {
                VALUE "CompanyName", "Microsoft Corporation"
                VALUE "FileDescription", "Directory Listing handler"
                VALUE "FileVersion", "10.0.18362.1 (WinBuild.160101.0800)"
                VALUE "InternalName", "dirlist.dll"
                VALUE "LegalCopyright", "© Microsoft Corporation. All rights reserved."
                VALUE "OriginalFilename", "dirlist.dll"
                VALUE "ProductName", "Internet Information Services"
                VALUE "ProductVersion", "10.0.18362.1"
        }
}

BLOCK "VarFileInfo"
{
        VALUE "Translation", 0x0000 0x04B0
}
}
```

Figure 22: Group 5 VERSIONINFO resource (left) mimics legitimate `dirlist.dll` module (right).

3.3.2 C&C communication

Few samples used encryption for C&C communication; however, Group 7 uses an interesting technique of embedding C&C communication in a fake PNG file, in an attempt to blend into regular network traffic. Furthermore, Group 11 uses DNS TXT records to obtain configuration data from the C&C server (via the `DnsQuery_A` API), to make the request look less suspicious, as shown in Figure 23.

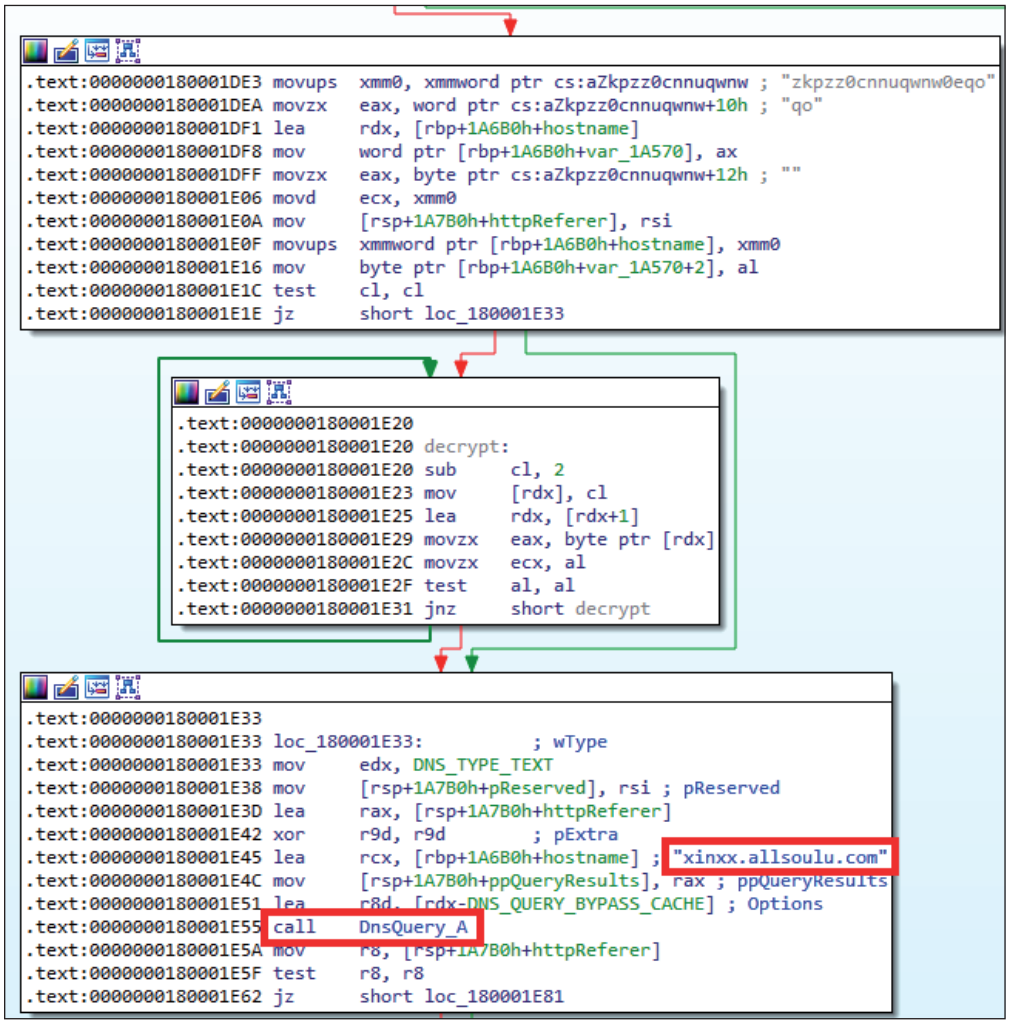

Figure 23: Group 11 uses DNS records to obtain its configuration.

3.3.3 Anti-logging features

The most notable evasion techniques used by the *IIS* malware families we analysed are measures to prevent the attacker requests from being logged on the compromised server, and thus to hide traces of malicious activities.

As *Palo Alto Networks* researchers demonstrated in their RGDoor blog post [9]: with default settings, the `Cookie` header is not logged on *IIS* (because it may be large and contain sensitive information). Group 4 (RGDoor), as well as some variants of Group 1, embed backdoor commands in the `Cookie` header.

Moreover, Group 7 uses a technique to prevent the server from logging attacker requests, regardless of the server settings. It implements the `OnLogRequest` handler, which will be called as part of the request-processing pipeline, just before the *IIS* server logs a processed HTTP request. If the malware detects a request from the attacker, this handler will 'sanitize' the log entry:

- It rewrites the HTTP method in the request to GET
- It rewrites the resource from the request to /
- It deletes these headers from the request: `Cookie`, `Origin`, `Referer`, `Sec-Fetch-Mode`, `Sec-Fetch-Site`, `Content-Type`, `Content-Length`, `X-Forwarded-IP`, `X-Forwarded-For`, `X-Forwarded-By`, `X-Forwarded-Proto`

Part of this handler is shown in Figure 24.

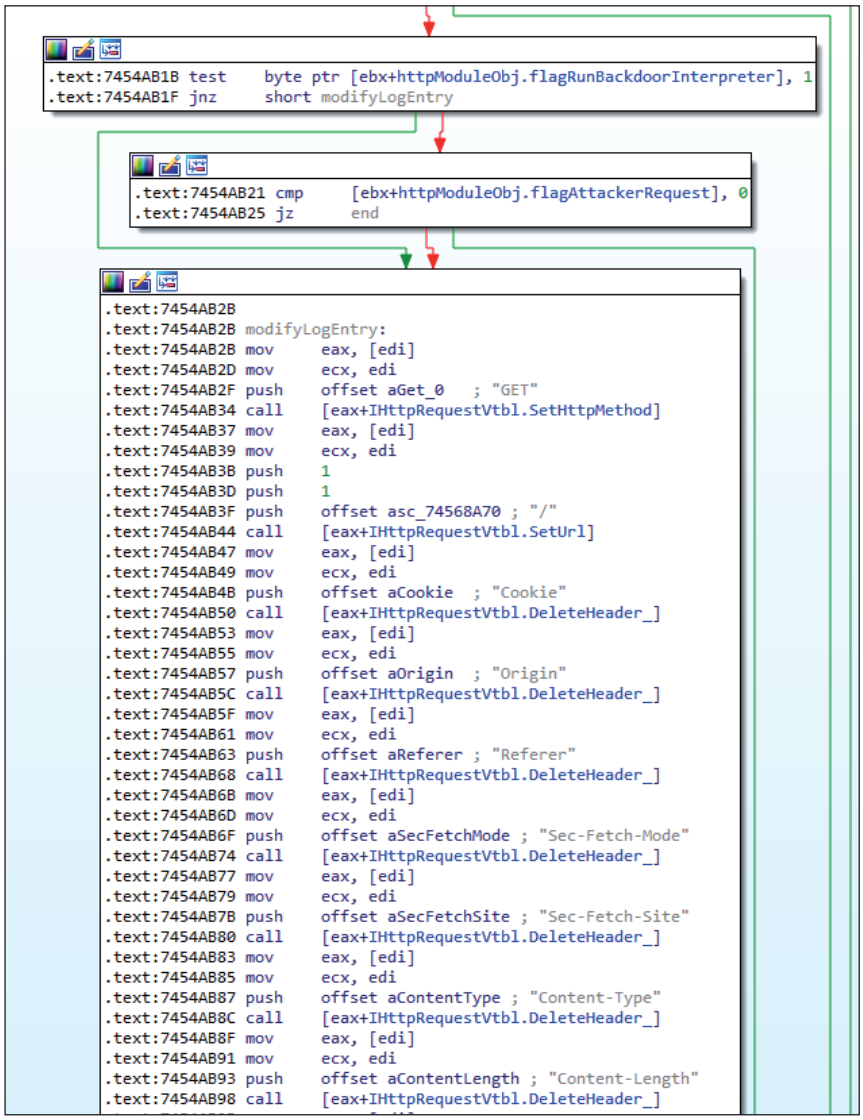

Figure 24: Group 7 modifies log entries for attacker requests.

3.4 Summary table

Table 4 (on the following page) summarizes key features of the analysed *IIS* malware families. For detailed analyses of Groups 1 – 14, please refer to the extended version of this paper [13].

4. MITIGATION

In this part we discuss measures that could be used to prevent the compromise of *IIS* servers, and how to navigate the *IIS* server to detect and remove native *IIS* malware.

4.1 Preventing compromise of IIS servers

Since native *IIS* modules can only be installed with administrative privileges, the attackers first need to obtain elevated access to the *IIS* server. The following recommendations could help make their work harder:

- Use dedicated accounts with strong, unique passwords for the administration of the *IIS* server. Require MFA for these accounts. Monitor the usage of these accounts.
- Regularly patch your OS, and carefully consider which services are exposed to the Internet, to reduce the risk of server exploitation.
- Consider using a web application firewall, and/or endpoint security solution on your *IIS* server.
- Native *IIS* modules have unrestricted access to any resource available to the server worker process; you should only install native *IIS* modules from trusted sources to avoid downloading their trojanized versions. Be especially aware of modules promising too-good-to-be-true features such as magically improving SEO.

Group #	Functionality					Attacker request verification (e.g. specific header present, specific URI, query string parameter)	C&C channel		Detection evasion and obfuscation techniques
	Backdoor	Infostealer	Proxy	SEO fraud	Injector		Encryption/ encoding	Alternative channel protocol	
Group 1	✓	✓	✗	✗	✗	HTTP header with hard-coded password	Base64	✗	✗
Group 2	✓	✗	✗	✗	✗	HTTP header with hard-coded password	RSA + AES-CBC	✗	✗
Group 3	✓	✗	✗	✗	✗	HTTP header present	Base64	✗	✗
Group 4	✓	✗	✗	✗	✗	HTTP header with hard-coded password	XOR + Base64	✗	Anti-logging
Group 5	✗	✓	✗	✗	✗	URI and HTTP header with hard-coded password	✗	✗	String stacking
Group 6	✗	✓	✗	✗	✗	Query string parameter	✗	✗	✗
Group 7	✓	✗	✗	✗	✗	Relationship between HTTP headers, HTTP body format	AES-CBC	✗	Anti-logging
Group 8	✓	✗	✗	✗	✗	HTTP header with hard-coded password	✗	✗	✗
Group 9	✗	✗	✓	✓	✗	No support for attacker requests	✗	HTTP	Encrypted strings (XOR 0x56)
Group 10	✗	✗	✗	✓	✗	No support for attacker requests	✗	HTTP to obtain JavaScript config	✗
Group 11	✓	✗	✓	✓	✓	HTTP header with hard-coded password	✗	DNS TXT to obtain config, HTTP for C&C	String encryption (ADD 0x02)
Group 12, variant A	✓	✗	✓	✓	✓	HTTP header with password whose MD5 hash is hard coded	✗	HTTP	String encryption (ADD 0x01)
Group 12, variant B	✓	✗	✗	✓	✓		✗	HTTP	UPX packing
Group 12, variant C	✗	✗	✗	✓	✗	No support for attacker requests	✗	HTTP	String encryption (XOR 0x0C)
Group 13	✓	✗	✗	✓	✗	Query string parameter	✗	HTTP	✗
Group 14	✗	✗	✗	✓	✓	No support for attacker requests	✗	HTTP	✗

Table 4: Summary of obfuscations implemented, and functionalities supported by analysed IIS malware families.

- Regularly check the `%windir%\system32\inetsrv\` and `%windir%\SysWOW64\inetsrv` folders to verify that all the installed native modules are legitimate (signed by a trusted provider, or installed on purpose).

For web developers: if you don't have the control over the *IIS* server where your web service is hosted, these measures can't be applied by you. However, you can still take steps to reduce the impact on users of your web service in the case of a compromise, especially:

- Do not send credentials to the server (not even over SSL/TLS); use cryptographically strong one-way salted hashes on the client side. *IIS* infostealers are a good example why server-side hashing is not good enough.
- Avoid unnecessary sending of sensitive information from the web application; use payment gateways.

Please refer to *OWASP* [39], [40] for more comprehensive information on secure web development practices.

4.2 Detecting compromised IIS servers

All native *IIS* modules are installed in the `%windir%\system32\inetsrv\` or the `%windir%\SysWOW64\inetsrv` folder. To check whether your *IIS* server has been compromised with native *IIS* malware, verify that all the installed modules are legitimate, using these methods:

- Verify the modules are signed by trusted providers.
- Use the IoCs listed at the end of this paper to look for suspicious modules.
- Use the YARA rules published on our *GitHub* repository [41] that we publish with this paper to search for Groups 1–14 analysed in this report.
- Use the free *ESET* online scanner [42] to reveal malicious modules.

Furthermore, check *IIS* server logs for indicators of malicious activity, as listed in the IoCs section. Pay attention to custom HTTP headers that attackers use to instruct their malicious *IIS* modules. To find the location of *IIS* server logs, open the *Internet Information Services (IIS) Manager* to find the *Logging* tab, as shown in Figure 25, or read it from the configuration file [43] `%windir%\system32\inetsrv\config\ApplicationHost.config`.

By default, the log files are stored under `%SystemDrive%\inetpub\logs\LogFiles` on the *IIS* server.

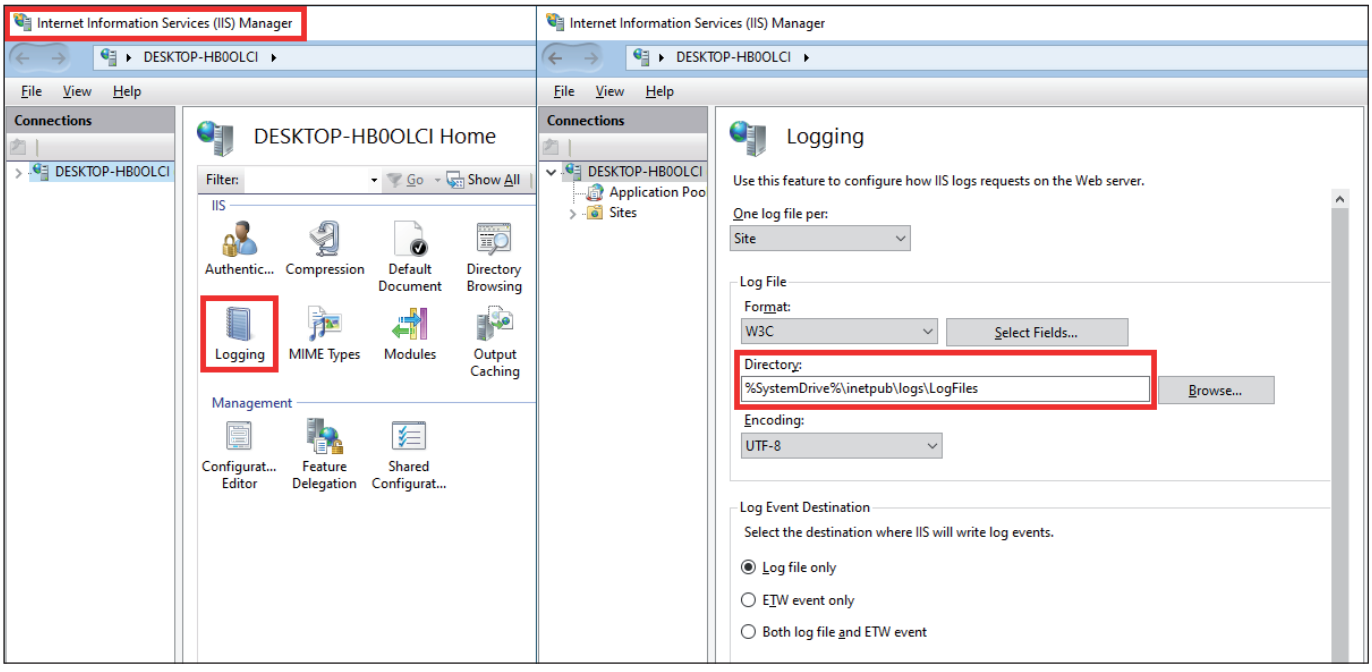

Figure 25: The log folder location can be found in Internet Information Services Manager.

Note that the *IIS* software is a built-in *Windows* feature that can be turned on with administrative privileges, even on desktop machines not intended as web servers [44]. Any *Windows* malware running with administrator privileges could enable *IIS* and use the compromised computer as a proxy (or for other purposes). If you find *IIS* running[8] unexpectedly, you can use the steps outlined above to verify that there are no malicious native *IIS* modules installed.

[8] That is, for example, if the *World Wide Web Publishing Service* service is present, or *IIS Worker Process* (`w3wp.exe`) is found running, but the details depend on the OS and *IIS* versions.

4.3 Removing native IIS malware

To uninstall a malicious native *IIS* module, follow these steps:

- Remove the module from the list of *IIS* modules configured on the *IIS* server. It's not enough to remove it from all web applications; it also must be removed globally (see examples later).
- Delete the malicious DLL file from the `%windir%\system32\inetsrv` or the `%windir%\SysWOW64\inetsrv` folder.

The module can be removed manually by editing the *IIS* configuration, via the *IIS Manager* GUI, or via the `AppCmd.exe` command-line tool [7].

Figure 26 illustrates how to remove an *IIS* module via *IIS Manager*. Select the *Modules* tab (1) and navigate to the module name (2) – in this example, `IIS Backdoor` installed under `C:\Windows\System32\inetsrv\httpaxd.dll`. Remove the module from the list of modules installed at the server level (3), and from the list of configured native modules (4-5).

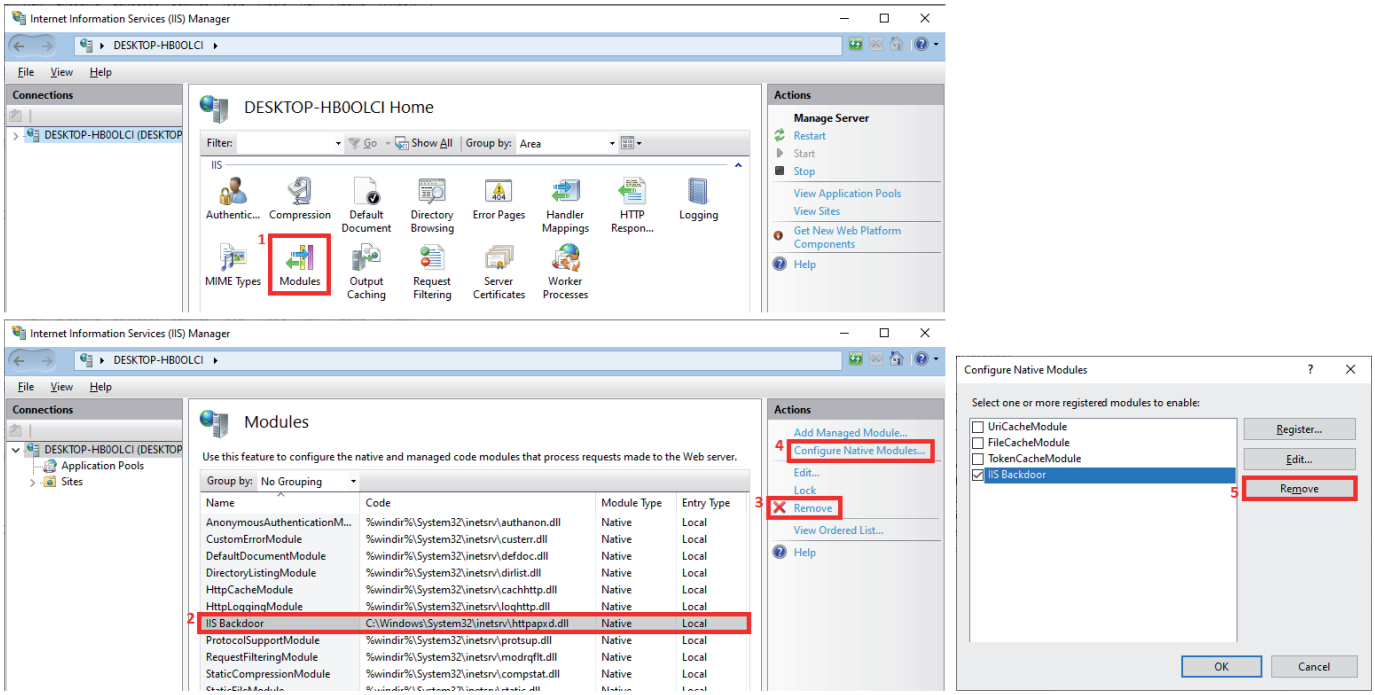

Figure 26: Removing a native IIS module using IIS Manager.

The equivalent action can be performed using the command-line tool `AppCmd.exe` (as also shown in Figure 27).

```
%windir%\System32\inetsrv\AppCmd.exe uninstall module <moduleName>
```

Administrative privileges are required for this action.

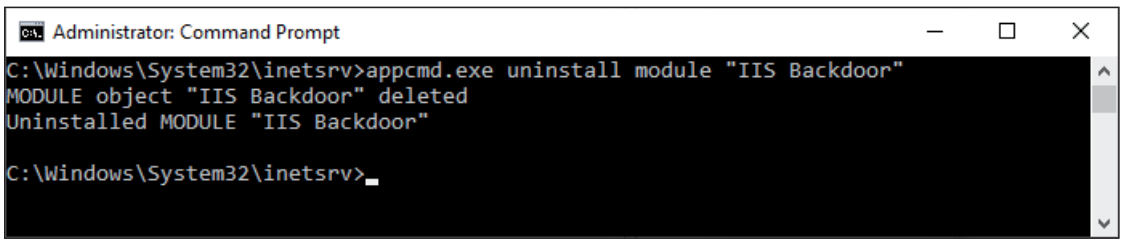

Figure 27: Removing a native IIS module using the `AppCmd.exe` tool.

It is not necessary to restart the *IIS* server to remove a module; however, the module itself may not be the only malicious component on the server. If you do not plan to reinstall the *IIS* server, it is highly recommended to scan for (and remove any) additional malware, make sure the OS and software are up to date, and modify the passwords of all the accounts that have administrative rights on the compromised server: otherwise, the attackers could reinstall the malicious *IIS* module.

5. CONCLUSION

Internet Information Services web servers have been targeted by various malicious actors, for cybercrime and cyber espionage alike. The software's modular architecture, designed to provide extensibility for web developers, can be a useful tool for the attackers to become a part of the *IIS* server and intercept or modify its traffic.

In our survey of the current *IIS* threat landscape, we collected and examined 14 native *IIS* malware families, most of them previously undocumented. Overall, the level of sophistication was low, but we did see evolution and some tricks that could challenge defenders.

Moreover, it is quite rare for endpoint (and other) security software to run on *IIS* servers, which makes it easy for attackers to operate unnoticed for long periods of time. This should be disturbing for all serious web portals that want to protect their visitors' data, including authentication and payment information. Organizations that use *OWA* should also pay attention, as it depends on *IIS* and could be an interesting target for espionage.

Admin privileges are required to install native *IIS* modules, but we found cases where attackers shipped the malicious malware as a trojanized *IIS* module, or spread *IIS* malware using server exploitation. The March 2021 mass-exploitation of the ProxyLogon vulnerability was only one example of how *IIS* malware can get to interesting data (in that case, government mailboxes).

While *IIS* server threats are not limited to native *IIS* malware, we believe this paper will be a helpful starting point for defenders for understanding, identifying and removing native *IIS* malware, and a guide for our fellow researchers to reverse-engineering this class of threats and understanding their common tactics, techniques and procedures.

ACKNOWLEDGEMENTS

We acknowledge Marc-Étienne Léveillé and Mathieu Tartare for their work on this investigation.

REFERENCES

[1] IIS. https://www.iis.net/.

[2] Grunzweig, J. The Curious Case of the Malicious IIS Module. Trustwave. 2013. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/the-curious-case-of-the-malicious-iis-module/.

[3] MITRE CVE. CVE-2021-26855. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-26855.

[4] MITRE CVE. CVE-2021-26857. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-26857.

[5] MITRE CVE. CVE-2021-26858. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-26858.

[6] MITRE CVE. CVE-2021-27065. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-27065.

[7] Volodarsky, M. IIS Modules Overview. Microsoft. 2007. https://docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/iis-modules-overview.

[8] Bureau, P-M.Malicious Apache module used for content injection: Linux/Chapro.A. ESET. 2012. https://www.welivesecurity.com/2012/12/18/malicious-apache-module-used-for-content-injection-linuxchapro-a/.

[9] Falcone, R. OilRig uses RGDoor IIS Backdoor on Targets in the Middle East. Unit 42. 2018. https://unit42.paloaltonetworks.com/unit42-oilrig-uses-rgdoor-iis-backdoor-targets-middle-east/.

[10] Secpulse. 万万没想到，我还是没辜负客户的期待. 2019. https://www.secpulse.com/archives/113500.html.

[11] TeamT5. 打倒夢魘！電子商務安全不再沉睡！！信用卡與個資竊取案例分析. 2020. https://teamt5.org/tw/posts/e-commerce-security-a-case-study-on-credit-card-data-breaches/.

[12] Faou, M.; Tartare, M.; Dupuy, T. Exchange servers under siege from at least 10 APT groups. ESET. 2021. https://www.welivesecurity.com/2021/03/10/exchange-servers-under-siege-10-apt-groups/.

[13] Hromcová, Z. Anatomy of Native IIS Malware. ESET. 2021. https://www.welivesecurity.com/2021/08/06/anatomy-native-iis-malware/.

[14] Netcraft. March 2021 Web Server Survey. 2021. https://news.netcraft.com/archives/category/web-server-survey/.

[15] W3Techs. Usage statistics of web servers. 2021. https://w3techs.com/technologies/overview/web_server.

[16] Microsoft Press. Internet Information Services (IIS) 7.0 Resource Kit. ISBN: 9780735624412, 2008. https://www.microsoftpressstore.com/articles/article.aspx?p=2231761.

[17] DEVCORE. ProxyLogon: The latest pre-authenticated Remote Code Execution vulnerability on Microsoft Exchange Server. 2021. https://proxylogon.com/.

[18] Nataraj, R. New Lemon Duck variants exploiting Microsoft Exchange Server. Sophos, 2021. https://news.sophos.com/en-us/2021/05/07/new-lemon-duck-variants-exploiting-microsoft-exchange-server/.

[19] Microsoft. PFN_REGISTERMODULE Function. https://docs.microsoft.com/en-us/iis/web-development-reference/native-code-api-reference/pfn-registermodule-function.

[20] Volodarsky, M. Develop a Native C\C++ Module for IIS 7.0. Microsoft. 2007. https://docs.microsoft.com/en-us/iis/develop/runtime-extensibility/develop-a-native-cc-module-for-iis.

[21] Microsoft. CHttpModule Class. https://docs.microsoft.com/en-us/iis/web-development-reference/native-code-api-reference/chttpmodule-class.

[22] Microsoft. CGlobalModule Class. https://docs.microsoft.com/en-us/iis/web-development-reference/native-code-api-reference/cglobalmodule-class.

[23] Templin, R. Introduction to IIS Architectures. Microsoft. 2007. https://docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/introduction-to-iis-architecture#request-processing-in-iis.

[24] Microsoft. IHttpModuleFactory Interface. https://docs.microsoft.com/en-us/iis/web-development-reference/native-code-api-reference/ihttpmodulefactory-interface.

[25] Microsoft. Request-Processing Constants. https://docs.microsoft.com/en-us/iis/web-development-reference/native-code-api-reference/request-processing-constants.

[26] Microsoft. IHttpContext Interface. https://docs.microsoft.com/en-us/iis/web-development-reference/native-code-api-reference/ihttpcontext-interface.

[27] Microsoft. IIS Server Variables. https://docs.microsoft.com/en-us/previous-versions/iis/6.0-sdk/ms524602(v=vs.90).

[28] Wikipedia. Payment gateway. https://en.wikipedia.org/wiki/Payment_gateway.

[29] MITRE ATT&CK. Proxy: External Proxy. https://attack.mitre.org/versions/v9/techniques/T1090/002/.

[30] MITRE ATT&CK. Proxy: Internal Proxy. https://attack.mitre.org/versions/v9/techniques/T1090/001/.

[31] Cherepanov, A.; Lipovsky, R. GreyEnergy: Updated arsenal of one of the most dangerous threat actors. ESET. 2018. https://www.welivesecurity.com/2018/10/17/greyenergy-updated-arsenal-dangerous-threat-actors/.

[32] Kaspersky GReAT. The Duqu 2.0: Technical Details. 2015. https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07205202/The_Mystery_of_Duqu_2_0_a_sophisticated_cyberespionage_actor_returns.pdf.

[33] O'Connor, P. An Introduction to Black Hat SEO. Hubspot. https://blog.hubspot.com/marketing/black-hat-seo.

[34] Google. Doorway pages. https://developers.google.com/search/docs/advanced/guidelines/doorway-pages.

[35] Google. Irrelevant keywords. https://developers.google.com/search/docs/advanced/guidelines/irrelevant-keywords.

[36] Google. Webmaster Guidelines. https://developers.google.com/search/docs/advanced/guidelines/webmaster-guidelines.

[37] Google. Cloaking. https://developers.google.com/search/docs/advanced/guidelines/cloaking.

[38] MITRE ATT&CK. Drive-by Compromise. https://attack.mitre.org/versions/v9/techniques/T1189/.

[39] OWASP. OWASP Secure Coding Practices-Quick Reference Guide. https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content.

[40] OWASP. OWASP Proactive Controls. https://owasp.org/www-project-proactive-controls/.

[41] https://github.com/eset/malware-ioc/tree/master/badiis.

[42] ESET's Free Online Scanner. https://www.eset.com/ca/home/online-scanner/.

[43] Microsoft. Configuration Reference <configuration>. 2016. https://docs.microsoft.com/en-us/iis/configuration/.

[44] Microsoft. Installing IIS 7 on Windows Vista and Windows 7. 2007. https://docs.microsoft.com/en-us/iis/install/installing-iis-7/installing-iis-on-windows-vista-and-windows-7.

INDICATORS OF COMPROMISE (IoCS)

ESET detection names

- Win32/BadIIS
- Win32/Spy.IISniff
- Win64/BadIIS
- Win64/Spy.IISniff.A

Samples

SHA-1	SHA-256	Malware family
1176B51814B59526B02AA7F3C88334A0256D0370	241CCE3BDE9379FCB18C81A856E8B675 82B44E48F2134B3E750A9370BD87D707	Group 1 (IIS-Raid derivates)
58E23A74AC78D223505C774A20D0C1DE1070BE3A	82B107AA1791A58C65E3A266981E886E 24E7D6ABCF076750F3E72BF4E3697AA4	
66739373D34DE7002C80A5E2AF660E9EA6FF6E7F	165B55F40E9B488A20A7346D7B110729 F0B7025CA767B0E174C6B45C9E09B42B	
68B6E7A48CD9B894A076C4D10A64E98F1C7366A3	4995992852F5C5066581E4D63BCB9B26 55D7458534C2710276587159C234E135	
6AC1CB3E04E45894DFFFC5D113068B28923508AA	68A57727AA0097CFE65782961FD10C8F 6FC8766C7A816F85098C3DDCF7A681BE	
77B9E64F4C2B7DF6AAC00D21A5EB071EEFC25596	7C61C37356A2486CB39EEF47FFB91B0E 64EFD2D1CAA9E686AE2422DFBA621E6D	
8A0885AE43FA9057A46611E8171EA2DDC5DA7330	364BF510C9C1D54EEDBCFAB6260E1EA7 2D9BCE0C9C8DD4EA8E84E254F5C1C91C	
8B8BBF897C49A01D82610F3834CC77111D310161	BCDE2EE839D6BC2E18BD150F4FC21BEB 369F7877425EFEB6E721BF954F54679D	
9DFA1EC8704A697F9847C0B8DFF41BF5EE289FC1	C9A69B28D4505B608DA384E104A9046D 1200AA84157ADC2DD1628C823F2C6323	
AA9BA493CB9E9FA6F9599C513EDBCBEE84ECECD6	A11626D55EE9C958D86E8C77DFE19F66 CDF545FBD8743126081F46DC24446767	
B24E67BDB53B380AC97249B489AADE3BFDAF9E43	D15D07ADD8F4F27AC87127C7D98D287F 7CD0A4E5D480119DAB62AA6488A70D59	
BBFC53367730C270C680EE5E7860F824102082C3	138A48279B17D4F04368096A6F2DE5D1 6CF3D4C4472342D3263468A69399B9B8	
BF2A9C216A1DA23BBDA004101A2E2F5E8D8D3D23	46644577E1D6F748A7C10667EED8255D E711B95018A4A75234F070409BA8BD8D	
C2F9740A73CCC13A868E1E3150F43BDDB54CC66A	44D95FF98BC70DCA8C9BAE7B7CDB2E6F 94685F5FC65F2EC5CD27D069C4E4797D	
ECAEA1D9D4E84FDDF61C87AB64256EAC7863D3A7	611A41A2856B907ABD2EBC627369AEBF E0156864F2927BB1A124A3DC1E8463A0	
F449C31AAB9EC0E6C6B2C336DEB83ADE6DAD53FD	17DE3F731A78BC740C5B57FB6D667CB6 8D93B5FE94076C852DDB30D7089988CC	
F8EF3168DD4C07D0C2E36D4143C9DDA8E1F6E306	2A2C1447A24FCA304815B8DE8B546276 E37A866F0BB9390A69F92EC150855A1F	
338B6E464874A52E61BC5B8FCAA94D66FE7E4141	0644B3FFC856EB54B53338AB8ECD22DD 005EE5AACFE321F4E61B763A93F82AEA	Group 2
481543A5985B947989691C01C478721AED5B0F2D	E733B9444106CA37C3EF9E207AC6C813 B787614496B275C1A455FCCC3ACA1C4A	
AB934E9A0BFCEFB2DF295E1E9ADBB3FFD1F15B82	CFAEC2A27DC9667443BC5BE81B66E01C 42AD5D83A90393E4DFFC396E46F99EE7	
E2EAA585E69150029487080E445E1240D918ED1D	9793EA98B7FBD43F0A7273594D7B4E53 338048C651C33FBFDBEB1CC275957996	
D33FA7C550AC0A7B47EB690FE9C3750CAF04EB68	F8EEB8A8E336EAA8723D483FD3DEC802 C504A7121976477A3A1D6BAF44F19A12	Group 3
5447283518473EA8B9D35424532A94E2966F7A90	497E6965120A7CA6644DA9B8291C6590 1E78D302139D221FCF0A3EC6C5CF9DE3	Group 4 (RGDoor)
A9143B0FC38B6329D5DFBFFC4AA91B5F57211DA0	A9C92B29EE05C1522715C7A2F9C54374 0B60E36373CB47B5620B1F3D8AD96BFA	
706EAB59C20FCC9FBC82C41BF955B5C49C644B38	D52EBFA1EA0366FFBCE967A652190E3E B0206E47319A19DF630D37443E7D0D69	Group 5
7A2FA07A7DC05D50FE8E201A750A3DC7F22D6549	7553767046F15E37550F3D26A779A7E8 EC3704842B97B928092602D725528A4D	
A1C5E7424E7C4C4C9902A5A1D97F708C6BB2F53A	95795DE242B0B42D4AD0BB66EF8D9BAA 0C2E9E35F419FB515F023E9A33ED271E	
A43D964E709EF8F7F035B85ED4AE9B26D4394B58	157174F0B9BE66E3C9090C95EFDD1DD2 3B19E42AA671758EBAC5540A173F760C	Group 6 (ISN)
E00E8477CEE2BEDA5B67346C9742C4002D6B567A	C6847600910AB196652A38E94ECF592E 645D43025D1D61B3710B2F715238307B	

SHA-1	SHA-256	Malware family
22F8CA2EB3AF377E913B6D06B5A3618D294E4331	9F7DE916E513F89E8B7192BFC1DAADF927110F3EAFA836D036FAD2B3DB1A93D7	Group 7
435E3795D934EA8C5C7F4BCFEF2BEEE0E3C76A54	00142E46C997FE7051AF5667953908CC876268BE30D61CD985F6265514639251	
CED7BC6E0F1A15465E61CFEC87AAEF98BD999E15	819500F6D820BFFD4290B172EB84721EEE9F4D3A5814D58A65D5A321CE3E51AB	
0AC34B71F1CBED482579509DDF12DC28312E11A5	3D482E87A0E97E70C8B2E7541BA0BDEE388029A5A7F26BBD62D981565CC3A91B	Group 8
1B9EC94251AD5E8F407584DC786B261CADD7FA8F	6549DD663B597A951781F1AB6B820079F4FFEEC85F396F349D5CE2F97B3F9BF6	
21618E3FE6D133403320B4430054394AED944105	6DA823FA4950B95F9ADA74E6899FB0A17C90E8F64C75BE43F037461F3EEE3D02	
36741260BDE2B9304302F5AEB63CAEB309979554	4FBDA60F74A4003BC93E75ACFFBD55520C99236052B527F920C67C18673E6BBB	
4F6EAD034BF9A0B27ECFF16A08DB3ED57EA9C7D4	15ADA077C7BD86103F729810EA33A4856BB2B39BA1C017293C492A347036C331	
528527C8166FC55C2D80824D7C94FD574EACD1BC	320DA8CC3E46DF550363D8A2452C2C459BF30142DA065CCB29A7F2B9629EE112	
54E98E2655B39DFA486A01A09AC4920B7639401D	40384F574E4EA0A1DD0876CF4AF60F79A4A0B37D2A8287A795B8AB5E3427521E	
5924800E432731C6699A80EF4D6AD9496EB1BF98	80E327564D00167A6EB4ECF5A6FA0526D5261B390C46EF442673C2E69173E470	
5A5545B868EC41A15810E9351ECA93110C878BC1	307F905981965AFC33BC17E5053D877DEB2EB4FE7B88B892D59DBAE96992C161	
8B0D9F3DBA9B05FFB91B8C77786B6FD85BEA6944	716C14EDBD08658FC72A7641913CBAB451C3F947D2473FD36488B1A228D1E340	
A70CA39BE949629C8EED1A71258ADB259E6A9D9E	C29E53DE6684D771CFF912A4AD57D203D1A63FF8334AA30727E76C874492AB54	
AED3625099606849755A6C25022D072BCE7E9EE7	6A9E2CB9592BEE7DDA6165FC5F8C26E6BE5EB49D9DE58B3251327C1D02D858B3	
D669F4DDE56DF8D032520399EDEA0F9971063F38	FDE34C06DFF9A5304C39409704723393 0DA199CCE90D5A2DED3A1634DEA42470	
E3FE87183E5F63A63915EB9B3740218A42CD6CF3	FEEEA74325D2F389E9325A8113F185BC823DC0681B86A82982C3A3F2951750C8	
279D841539212D4F159404417404827EBC17B8D7	2EA3202CA7EBD5C407409D35E71520F4782A136454487CA857AFB5032660F93F	Group 9, variant A
44933C61C82B9FB7C2A17B32C010C5B044B638F3	3150C48438D4781D4C3ADA83B7BE45D76D8AC7A78F5D8D602152AC1ABC3528BF	
4E8B84412101112E73F846545A412127AA5DCEB8	0CBC4D4941C509608C0892BF337ABE6B004A2FA7C1E83E7FF23D54E323064FAF	
52FDE6863D8C3E79913B29EFA656C3B32FE2CF45	C79086813D0C846DEECB7EECA238F78A662F0AEC1DED892C3561522CBB39A24A	
5B205F3C19CAA177C0D32EBCCCAA3D3202764132	A19144FAD371A7FB476E5C109E1CA943245C41EA833C5E10AD4FF0DB0E045869	
62C47260DC013DDF625F0016736576BDF4E3B212	F5615C120AAAB860B279E095A68EA0AE2CA556929395F118AD7B63AF53D61F21	
64DF8B5AD43A0C4D81DFF075898E492FBAF1CD9E	BBDADA0149CD4833A32E9F0D981E36ED13685B1F00233E7196B8432EC1589B3D	
7DA9FBD4BAB842DADB943790E69B0C15E74EC614	507B77AB91F1B9C792210D7E38F4D43F16AB652F2B3008F1361CACC81817F992	
93C40123D11EFC0C75F9C8E515EA49B4D047D8C1	CEFD1C9ABBFE0DD44D923A24A568B3531D067EF821F40CA64C471A8AA1FF33D3	
A3E64F4D0898B77E5AE931029BCD330F2694643E	0B7FED82D2594B8A30772EEC6EB6BF2DB6A23404504535BB78C828EC1FC870F5	
A41F73A3A28E46ADBD6753F9B0A005E8A4FA55FD	91BB5A365478DE474D938690F4EE9BCD6413EF59D331829DA93C9C1C88FCB771	
A42893843059ED9922FDAFFF0A02DF4F39519930	12F6B72CDF8660D94EB5D915D4EDDC0EE3AE4ADAA719CACAD60C6F7D44E90486	
B8051F1B51EC093BA56B1F70C8AE63EB6A9644C1	08EE575B9CDA0EA5F12C8D5132469C99CD1DEEBFC9514F7B8CB520348D3A9ABC	

SHA-1	SHA-256	Malware family
BD98ABC510AC3DF66E21C3CDCEE7BDFC1A326AC5	A62734619EC889E7C80BB2EDC3497CD4139EBD5646DB30C20E0928B264B53435	Group 9, variant A
FABCEFF3B03D3DA0B338E2EC0F7D47E83E86F720	E3ACE9E5B9A71A6A2E98DAA33FD19E536D2A520A0160495B4B77EC98AD0F71D9	
3A3DF03DA61F86CEFEB7A1546421346CE2608DE1	28B4C4F001517AC4A682B728FD4B9D1364DA6698A84D2F1F6016937E8240219E	Group 9, variant B
7080CC770A99FF57FD899E367B7F2A430FC55CD1	4F8775E26D6E291905F49A2766B804C6FE8398D8ACC26C11EC6A2FA96A02B3DE	
8A25CBCF5B7DEBCA8A9E55D233E816EA6FBE8A0F	7353030AF3274EA1AB9756AAAD8130FB01BCACD82FB6C6D2358DDCD060257275	
B626B9A712FCF957438DBED889575D3F9E1B33C8	9ABC210DE663EFA287E09ADFB6FE4196D46D4F3F4541FD4D64439ADC220709AE	
DD9F72DE1070903359ACEA98884A953B23CB7354	F0E95504B127DC1477CC1D89AAC29EDB8B7578CE21224DDB4D67C3B81AB19E52	
72FB52C21EDC50D79DDDAAAE6EE473713CE4F82D	9869782B98AFEF7B1619CEBBBE3A45EC4BC50C7138A4E8291A31F2E039D08B46	Group 9, variant C
2ED260A0EA017D0322C494E9EBFB0E1C07A6A0F2	56715F3E15E8D39125E0B8CB46AAAC1788FA46DF4EED6881DDC5BEB805679506	Group 10
33F999E9F31648B3115D314EC49B09A005EC992A	59249BEDE0DEAEA326C5BB6584DAF5E25C9F65EDE0AF7E7CD5F63761DD91B3AF	Group 11
A2EF7DE7A217B6F9F0F886B5D4DE4C56D5A6A7BE	FB07C5B6E8F0AE482D9C571611F5868179227938E1E23DE3D09DCBCB14FB7972	
1E82C6DB2EE1688BF8B182FF93C9BEA8CD84BEC1	CB816863576B982FB7F14A41C63282D8F6F7A635E555353F5A75110794196F87	Group 12, variant A
7554B6A5244D4BCE83C2ABE762174399CDE1ED05	AD1C768F5F6BBA0110BE23C36AD6AAFFCA7F122CF3A5624934AF6CF871F58BEE	
DCACD46E441C42AA0EACBC99F072F3BA6A91D02B	41CD5131C323ED643BEBB245DA7EC39F49EFE1014BCE2F3B4031BA5903FB97DB	
09BFC4596BAEF62BD2FFA79D5A4D4116B0186DB5	4D4BCF6BE29B3074D01F839D81A78880BE7AFC5DF366D65006A5D07FC9D11FEA	Group 12, variant B
6C531446598E743D315A74B4C1B3C08BE2B70C0C	B8E15597E1B137274F36C5E5F6F0811F041DBD5C2CD0784A2A928F6EEB68CBA2	
AFB59D38754ED71B251F89A999ECFB2046D5CB21	3AE3AE44712A4CC7645BBCE3B54F6A431EA08D33105F78CD8F330027AA15B8CC	
FA790BCC0338899ABBF6C573D7FB76086A8CD62D	2F708F00F6C2743F61B662DCC82AC908F5D86C6A87D72CC7061311D267D36E56	
5A3CC5E97AD448BF3DDDD4ABFD59BC74CD23B583	05D2FDE8B6141318A97CB0044C2494F009761351AF9B6633ECC7E7F089879998	Group 12, variant C
D0F274EBD2A0636FEF9D9C48A7AC2FAD7B661653	AA34ECB2922CE8A8066358A1D0CE0FF632297037F8B528E3A37CD53477877E47	Group 13
086A211A069322DF84484E0E4B4B4D8AF3ADE95B	CDB7C3638FFFFFD42111E0A72DC959F1B49E15BE7E8BB9A7BAD2C5D89CC00F8B	Group 14
30BE5F13FA182008EBE991C0795AFD3783AAA903	01830EA1E8BCFA8307D1D271982EF40C3451A21F7B109835B524F7A2F5F50DCC	
CD1B29BFD41F469D9CB25FA282F26B3B2AB422B9	64A2785B41C0864CB630F54D749371E4AE6D916D421B96F0E394D203C066C883	

Filenames and paths

Malicious IIS module names

Note: All of these files are installed under `%windir%\system32\inetsrv` or `%windir%\SysWOW64\inetsrv`.

Group 1

```
authmd4.dll
cachport.dll
httpapxd.dll
iiscom.dll
IISNET4.dll
IIS-Raid-Backdoor.dll
IIS-Trojan.dll
```

```
IISModule.dll
Microsoft.Exchange.Clients.OwaAuth.dll
svcfilter.dll
```

Group 2

```
iisddos.dll
iisred.dll
iissup.dll
```

Group 3

```
statdoc.dll
```

Group 4

```
HTTPParser.dll
TrafficHandler.dll
```

Group 5

```
dir.dll
isapicache___.dll
isapicache_.dll_
```

Group 6

```
iis7_32.dll
iis7_64.dll
```

Group 7

```
cache.dll
logging.dll
```

Group 8

```
FilterSecurity32.dll
FilterSecurity64.dll
iiscrash32.dll
iiscrash64.dll
```

Group 9

```
autehbas.dll
autohbas.dll
dirshow.dll
httpevt.dll
mscorevt.dll
sortkey.dll
webdac.dll
windows.dll
```

Group 10

```
FilterSecurity.dll
```

Group 11

```
HttpCache.dll
httpuser.dll
ispric.dll
```

Group 12

```
authcutd.dll
ManagedEngineV4.1_32bit.dll
ManagedEngineV4.1_64bit.dll
mscore.dll
```

Group 13

```
stati.dll
```

Group 14

```
urlresol.dll
```

Log file names

```
C:\Windows\Temp\AAD30E0F.tmp
C:\Windows\Temp\creds.db
C:\Windows\Temp\log.tmp
C:\Windows\Temp\thumbs.db
DllResolve.db
C:\Windows\Temp\cache.txt
```

Network indicators

HTTP headers combinations

Group 1

```
COM_InterProt, X-Chrome-Variations
Cookie, X-FFEServer
Sense-Pwd, X-Chrome-Variations
Strict-Transport-Security, X-Content-Type-Options
X-BLOG, X-BlogEngine
X-Cache, X-Via
X-Password, X-Chrome-Variations
XXXYYY-Ref, X-Chrome-Variations
```

Group 5

```
X-IIS-Data
```

Group 8

```
Cmd
```

Group 11

```
Cmd
```

Group 12

```
3389, Cmd
```

C&C servers

Group 9

```
http://20.3323sf[.]com
http://20.3323sf[.]com/zz.php
http://bj.whtjz[.]com
http://bj.whtjz[.]com/zz1.php
http://bj2.wzrpx[.]com
http://bj2.wzrpx[.]com/zz1.php
http://cs.whtjz[.]com
http://cs.whtjz[.]com/zz.php
http://df.e652[.]com
http://df.e652[.]com/zz1.php
http://dfcp.yyphw[.]com
http://dfcp.yyphw[.]com/zz1.php
http://es.csdsx[.]com
http://es.csdsx[.]com/zz.php
http://hz.wzrpx[.]com/pq.php
http://hz.wzrpx[.]com/zk.php
http://id.3323sf[.]com/wh1.php
http://id.3323sf[.]com/zid.php
http://qp.008php[.]com
http://qp.008php[.]com/zz.php
http://qp.nmnsw[.]com
http://qp.nmnsw[.]com/zz1.php
http://sc.300bt[.]com
http://sc.300bt[.]com/zz.php
http://sc.wzrpx[.]com
http://sc.wzrpx[.]com/zz1.php
```

```
http://sf2223[.]com/xin.html
http://sx.cmdxb[.]com
http://sx.cmdxb[.]com/zz1.php
http://sz.ycfhx[.]com
http://sz.ycfhx[.]com/zz1.php
http://xpq.0660sf[.]com
http://xpq.0660sf[.]com/zy.php
http://xsc.b1174[.]com
http://xsc.b1174[.]com/zz1.php
```

Group 10

```
https://js.breakavs[.]com/93/jc.js
```

Group 11

```
143.92.48[.]38
http://www.allsoulu[.]com
xinxx.allsoulu[.]com
```

Group 12

```
http://ee.allsoulu[.]com
http://202.100.206[.]136:443
http://center.g666[.]org:443
http://m.goudie[.]in:1024
http://m.goudie[.]in:1024/?zz
http://m.goudie[.]in:1024/js.html
http://m.pz8[.]in
http://www.g666[.]org/pic
http://www.pz9[.]in
http://tz.allsoulu[.]com
http://www.allsoulu[.]com
http://zz.allsoulu[.]com
```

Group 13

```
http://sb.qrfy[.]net
```

Group 14

```
now.asmkpo[.]com:80
speed.wlaspsd[.]com/vip.js
```