Backdoor Investigation and Incident Response: From Zero to Profit

Anthony Cheuk Tung Lai[1,2(✉)], Ken Wai Kin Wong[1,2], Johnny Tsz Wun Wong[2], Austin Tsz Wai Lau[2], Alan Po Lun Ho[2], Shuai Wang[1], and Jogesh Muppala[1]

[1] Hong Kong University of Science and Technology, Hong Kong, China
{lct,wkwongal}@connect.ust.hk
[2] VX Security Research Group (VXRL), Hong Kong, China

Abstract. We have investigated an incident in an online gaming company about an unauthorized access to the transactional database, the attacker modifies the gaming transaction to win the game. The attacker compromises the database occasionally without explicit footprints, basically this company has just engaged enterprise-grade firewall and anti-virus software but its anti-virus control failed to detect the existence of a backdoor file. After 4-month of investigation, with additional layered defense and monitoring, we have discovered the backdoor and carried out an incident response successfully. In view of this incident, OilRig attack [1] and Solarwinds Supply Chain Hack [2], we can foresee this type of incident will continue. Therefore, in this paper, we propose an incident response methodology matrix called BackDoor Incident Response Model (BDIRM) to handle incidents with backdoor effectively, thereby accelerating to eradicate the risk and impact of backdoor against organizations.

Keywords: Incident response · Backdoor · Malware · Targeted attack · APT

1 Background

Backdoor is a trigger-based malware, which generates traffic and its communication over authorized protocols (e.g., HTTP and DNS). Backdoor malware typically looks legitimate, which are usually ignored by network and system administrators.

We have carried out a security monitoring and incident response for an online gaming company. During our monitoring over the database system, we have found the modified transactions are executed by malicious stored procedures (see Fig. 1a and Fig. 1b). We have attempted to apply various industry popular incident response frameworks [3, 3]. The high-level methodology of incident response is comprehensive, and we can successfully identify and implement the incident response at the database server. However, it failed to detect and identify highly stealthy malware backdoor and attack origin with existing frameworks (Fig. 2). As a result, attackers keep getting into the system from time to time to make unauthorized transactions, even servers have been patched with up-to-date vulnerability and anti-virus definitions.

P. Gladyshev et al. (Eds.): ICDF2C 2021, LNICST 441, pp. 229–247, 2022.
https://doi.org/10.1007/978-3-031-06365-7_14

The online company has a basic infrastructure diagram for us to reference and change control of configuration and deployment details are inadequate.

We have found that a backdoor was installed to a Microsoft Web server for at least four months until we have established and implemented an incident response matrix to uncover this backdoor, which hits our checkpoints.

The revealed backdoor comprises functions to access a database and modify table entries of a transaction. The backdoor allows an attacker to interact with it to execute various built-in SQL query commands against the application and database and database system administrator (SA) process take-over statement (Fig. 1a) and create unauthenticated and authorized sensitive transaction records (Fig. 1b).

```
rdata:000000018005AB40 ; CHAR aAlterServerRol[]
rdata:000000018005AB40 aAlterServerRol db 'ALTER SERVER ROLE [processadmin] ADD MEMBER [%s]',0
rdata:000000018005AB40                                         ; DATA XREF: AlterServerProcessAdmin+42↑o
rdata:000000018005AB71                 align 8
rdata:000000018005AB78 ; CHAR aAlterServerRol_0[]
rdata:000000018005AB78 aAlterServerRol_0 db 'ALTER SERVER ROLE [processadmin] DROP MEMBER [%s]',0
rdata:000000018005AB78                                         ; DATA XREF: sub_180025540+42↑o
rdata:000000018005ABAA                 align 10h
rdata:000000018005ABB0 aSetNocountOnDe_2 db 'SET NOCOUNT ON;',0Dh,0Ah
rdata:000000018005ABB0                                         ; DATA XREF: sub_180025A60+77↑o
rdata:000000018005ABB0                 db 'declare sa_clear_cur cursor for',0Dh,0Ah
rdata:000000018005ABB0                 db 'select spid from sys.sysprocesses where loginame = ',27h,'sa',27h
rdata:000000018005ABB0                 db ' and hostname != ',27h,27h,0Dh,0Ah
rdata:000000018005ABB0                 db 'open sa_clear_cur',0Dh,0Ah
rdata:000000018005ABB0                 db 'declare @sa_spid int, @top1SQL varchar(max)',0Dh,0Ah
rdata:000000018005ABB0                 db 'fetch next from sa_clear_cur into @sa_spid',0Dh,0Ah
rdata:000000018005ABB0                 db 'while @@FETCH_STATUS = 0',0Dh,0Ah
rdata:000000018005ABB0                 db 'begin',0Dh,0Ah
rdata:000000018005ABB0                 db 9,'set @top1SQL = ',27h,'KILL ',27h,' + RTRIM(@sa_spid)',0Dh,0Ah
rdata:000000018005ABB0                 db 9,'exec(@top1SQL)',0Dh,0Ah
rdata:000000018005ABB0                 db 9,'fetch next from sa_clear_cur into @sa_spid',0Dh,0Ah
rdata:000000018005ABB0                 db 'end',0Dh,0Ah
rdata:000000018005ABB0                 db 'close sa_clear_cur',0Dh,0Ah
rdata:000000018005ABB0                 db 'deallocate sa_clear_cur',0Dh,0Ah,0
rdata:000000018005AD57                 align 8
rdata:000000018005AD58 ; const CHAR byte_18005AD58
```

Fig. 1a. Backdoor attempts to alter the privileges of the system and take over system process privilege.

```
.rdata:000000018005A1A0 ; CHAR aSetNocountOnIf[]
.rdata:000000018005A1A0 aSetNocountOnIf db 9,9,'SET NOCOUNT ON',9,9,'if (object_id(',27h,'tempdb..##dynupper',27h
.rdata:000000018005A1A0                                        ; DATA XREF: sub_180022B50+B6↑o
.rdata:000000018005A1A0                 db ') is null)',9,9,9,'begin',9,9,9,'return;',9,'end',9,9,'declare @U'
.rdata:000000018005A1A0                 db 'sername varchar(MAX) = ',27h,'%s',27h,';',9,9,'declare @MTime int'
.rdata:000000018005A1A0                 db ' = %d',9,9,'declare @tmpCSAgent table(csid int)',9,9,'declare @tm'
.rdata:000000018005A1A0                 db 'pTmpID int = -1',9,9,'declare @MAXFOR int = 0',9,9,'set @tmpTmpID'
.rdata:000000018005A1A0                 db ' = (select top 1 ParentCSID from Account where Username = @Userna'
.rdata:000000018005A1A0                 db 'me)',9,9,'insert into @tmpCSAgent values(@tmpTmpID)',9,9,'REPER:',9
.rdata:000000018005A1A0                 db 'set @tmpTmpID = (select top 1 Parent from CSAccount where ID = @t'
.rdata:000000018005A1A0                 db 'mpTmpID)',9,9,'if @tmpTmpID != -1 and @MAXFOR < 1',9,9,9,'begin',9
.rdata:000000018005A1A0                 db 9,9,'insert into @tmpCSAgent(csid) values(@tmpTmpID)',9,9,9,'set @'
.rdata:000000018005A1A0                 db 'MAXFOR = @MAXFOR + 1;',9,9,9,'goto REPER;',9,9,9,'end',9,9,'else',9
.rdata:000000018005A1A0                 db 9,9,'begin',9,9,9,'if exists(select * from CSAccount where ID in(s'
.rdata:000000018005A1A0                 db 'elect csid from @tmpCSAgent) and DATEDIFF (mi, CSAccount.LastLogi'
.rdata:000000018005A1A0                 db 'n, GETDATE()) <= @MTime) ',9,9,9,9,9,'begin',9,9,9,9,'update ##dynu'
.rdata:000000018005A1A0                 db 'pper set st = 1 where un = @Username',9,9,9,9,'end',9,9,9,'else',9
.rdata:000000018005A1A0                 db 9,9,9,'begin',9,9,9,9,'update ##dynupper set st = 0 where un = @Us'
.rdata:000000018005A1A0                 db 'ername',9,9,9,9,'end',9,9,9,9,9,'end',9,9,9,9,'select * from ##dynu'
.rdata:000000018005A1A0                 db 'pper where un = @Username',0
.rdata:000000018005A552                 align 4
.rdata:000000018005A554 aSt_0           db 'st',0              ; DATA XREF: sub_180022B50+164↑o
.rdata:000000018005A557                 align 20h
.rdata:000000018005A560 ; CHAR aSetNocountOnDe[]
.rdata:000000018005A560 aSetNocountOnDe db 9,'SET NOCOUNT ON',9,'declare @Username varchar(MAX) = ',27h,'%s',27h
.rdata:000000018005A560                                        ; DATA XREF: sub_1800236F0+B6↑o
.rdata:000000018005A560                 db ';',9,'declare @MAXFOR int = 0',9,'declare @tmpCSAgent table(csid '
.rdata:000000018005A560                 db 'int)',9,'declare @tmpTmpID int = -1',9,'set @tmpTmpID = (select t'
.rdata:000000018005A560                 db 'op 1 ParentCSID from Account where Username = @Username)',9,'inse'
.rdata:000000018005A560                 db 'rt into @tmpCSAgent values (@tmpTmpID)',9,'REPER:',9,'set @tmpTmp'
.rdata:000000018005A560                 db 'ID = (select top 1 Parent from CSAccount where ID = @tmpTmpID)',9
.rdata:000000018005A560                 db 'if @tmpTmpID != -1',9,'begin',9,9,'insert into @tmpCSAgent (csid)'
.rdata:000000018005A560                 db ' values (@tmpTmpID)',9,9,'set @MAXFOR = @MAXFOR + 1; if @MAXFOR <'
.rdata:000000018005A560                 db ' 100 begin goto REPER end',9,'end',9,'select ID, CreateDate, User'
.rdata:000000018005A560                 db 'name, Name, LastLogin, LastLoginIP from CSAccount where ID  in (s'
.rdata:000000018005A560                 db 'elect csid from @tmpCSAgent);',9,9,0
.rdata:000000018005A7CA                 align 10h
```

Fig. 1b. Partial SQL queries to show Backdoor malware attempts to create fake transactions in temporary database table.

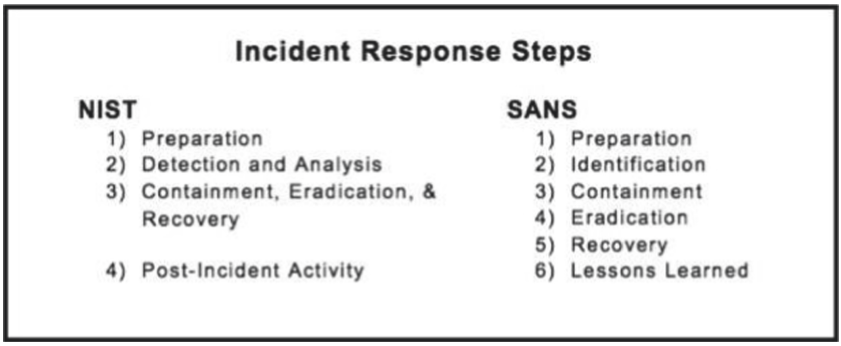

Fig. 2. Incident Response Steps from NIST [3] and SANS [4].

In a typical Web application system architecture, we engage three-tier design, and the data access layer is always deployed as a data access layer between a business logic layer in the Web application server and database [5]. It is not common to deploy a Web Server layer (before the presentation layer). In certain scenarios, a native extended module is developed to have database tables or logs backup, temporary transaction table clean up and different database maintenance tasks.

We have uploaded the backdoor malware named as Transtatic.dll to VirusTotal [6] which is a popular online malware scanner, it is found that all existing Anti-Virus software including Windows Defender, Symantec Antivirus and Firewall cannot detect the backdoor (Fig. 3a and 3b).

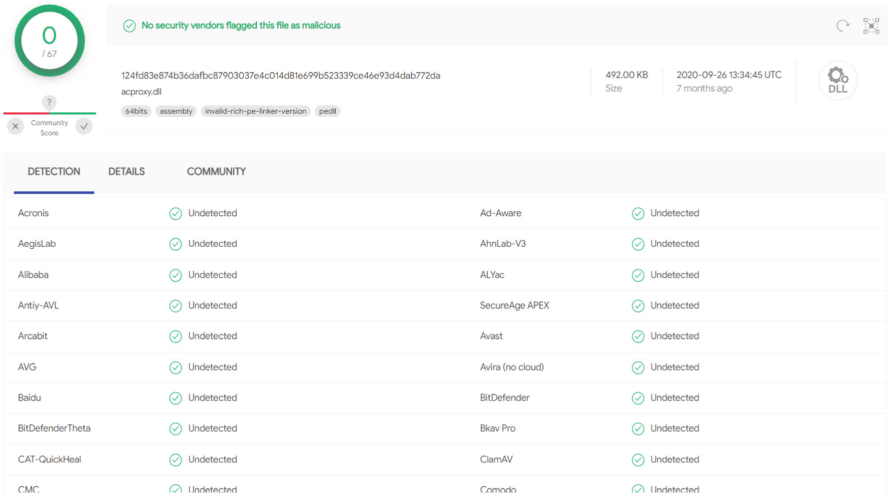

Fig. 3a. We have uploaded the backdoor DLL file with file name Transtatic.dll (SHA-256: 124fd83e874b36dafbc87903037e4c014d81e699b523339ce46e93d4dab772da) to VirusTotal and none of the anti-virus software can detect it.

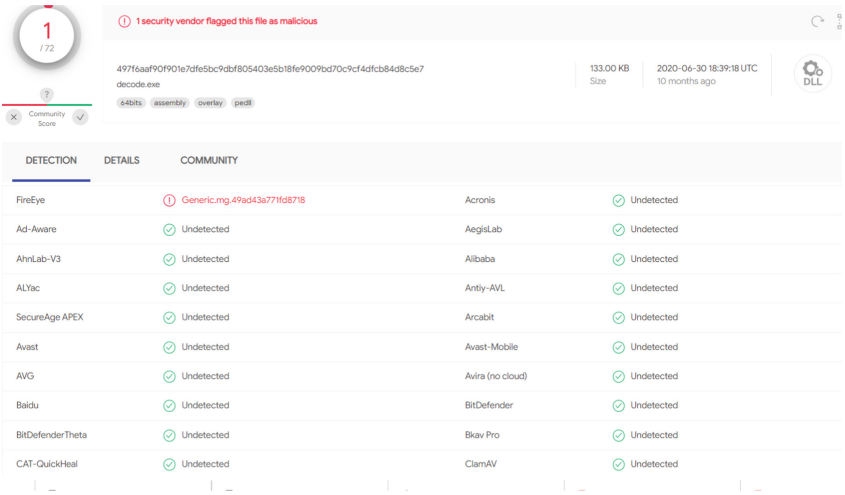

Fig. 3b. We have uploaded another published backdoor DLL file (SHA-256: 497f6aaf90f901e7dfe5bc9dbf805403e5b18fe9009bd70c9cf4dfcb84d8c5e7) from OilRig APT campaign to VirusTotal and there is one out of seventy-two anti-virus software can detect it.

In most of the target attacks, known as Advanced Persistent Threat (APT), which targets a specific type of industry and/or an organization, aiming for espionage, financial interest and service disruption. A Target Attack is well organized by a group of determined, well-focused and coordinated attackers [7] and deliver the attack in various stages including reconnaissance (also known as information gathering), attack vector delivery, exploit and compromise the target and maintain a persistent connection with the infected systems [8], backdoor serves the purpose of revisiting of the compromised system.

In view of the stealthy characteristic in the binary of and network traffic generated from backdoor, we hope to propose a novel incident response matrix specifically for Backdoor, which not only include binary executable backdoor but also html or php-based backdoor and Webshell.

2 Challenge in Analyzing Backdoor and Incident Response

2.1 Limitation of Behavioral Analysis

Most malware samples can be analyzed via behavioral (as known as dynamic) analysis, which means that we execute the malware and study the behavioral indicators. While it is applicable well to common malware samples, it is not applicable for our scenario. Overall, for the Web server backdoor, we need a particular setup of a server to load the native module, and once it is launched, there will be no active footprints made by the backdoor as it waits for the command from the attacker. Malware analysts should reverse engineer the backdoor before carrying out any interaction with the backdoor. Depending on the complexity of the backdoor sample, reverse engineering can be very resource-intensive and impractical to facilitate real-world incident response and detection.

2.2 Emerging Backdoor Threat

Our observation shows that an attacker inclines to believe that if a backdoor to the web server is installed, it becomes impossible for the security administrator to discover, given the web server traffic is usually high. The context of request headers can be varied a lot, and it is challenging to identify malicious requests. There are research works using machine learning or deep learning models to detect Malware threat in this context. However, we argue that in realistic enterprise environments, the application of machine learning and deep learning will notably delay the transactions and cause performance bottlenecks.

Even in the best case, the security administrator can identify the attack requests, due to web server log limitations, it only captures simple GET requests in URI. As backdoor attacks are always communicating via HTTP POST request instead of GET request, it is impossible to have a full diagnosis of the malicious activity. Moreover, there is a large target attack campaign named OilRig [1], so called "Advanced Persistent Threat (APT)" that has engaged the IIS native module as their backdoor to access victims' servers.

2.3 Lack of Practical Backdoor-Specific Incident Response Methodology

There are different industry-recognized incident response frameworks and methodologies. For example, NIST [3] and SANS [4] have suggested similar high-level methodology in conducting incident response as follows:

1. Preparation
2. Detection and Analysis
3. Containment, Eradication and Recovery
4. Post-Incident Activity

To analyze Malware, FIRST [9] suggests performing media and surface analysis, reverse engineering, dynamic analysis, and comparative analysis. MITRE Att&ck [10] proposes a more detailed breakdown to identify Command and Control communication. However, those are still falling to conceptual checkpoints.

We hope to extend the methodology in practical and experimented backdoor incident response areas with our real-life experience, which provides shortcuts to detect and identify backdoor efficiently and effectively.

3 Incident Response Model

We are now proposing a BackDoor Incident Response Model (BDIRM) with an algorithm to identify compromised hosts and detect and analyze backdoor with the matrix.

Illustrated in the following network graph (Fig. 4), We define the graph G as G = (V,E) where V denotes a set of servers, computers or accounts (v_1,v_2,v_3,…,v_N), named as *nodes*. E is a relationship set of connections (e_1,e_2,e_3,…,e_N), between pairs of nodes, named as *edges*. We label the blue line as the connection between two trusted systems/parties deemed by the organization. We label the connection as red line when there is a connection between a trusted system and another untrusted system or between two untrusted systems. We have defined the number of authenticated systems connected to the node v_i as Ns(v_i) and number of different user accounts used to authenticate to different systems as Nua(Ns(v_i)). To differing the different configuration c_i with a legitimate one in a node, we define it as Di[c_1,c_2,…,c_N].

Attackers attempt to take over each connection or trusted parties to become their steppingstone to reach out the final target, in this case is the database, blue and green nodes are trusted from the organization perspective. With this algorithm, we have detected and identified the backdoor incident effectively:

Algorithm 1. BackDoor Incident Response Detection Algorithm (BDIRDA)

For every node (v_i): start with the least number of trusted connections to the target node:

a. Examine user authentication and Delete-Create-Execute-Delete-Create operations or activities over the files/stored procedures/scripts/user account in the event or/and activity logs in different systems or/and application of the potential compromised host.

b. Examine the number of authenticated systems connected to the examined node (Ns(v_i)) AND examine the number of different user accounts used to authenticate to different systems (Nua(Ns(v_i))).
If the ratio Nua(Ns(v_i))/Ns(v_i) > 1, it is suspicious where a single node (v_i) is authenticated to many different systems with multiple different user accounts that are not in normal business practice.

c. Differ configuration files of any service available to untrusted parties with the intended and legitimate configuration Di[c_1,c_2,…,c_N].

d. Label the node(v_i) and edge (e_i) connected to and from any node (v_N) as red if any item from 1 to 3 is positive and suspicious.

If any red node (v_i) satisfies any of above checkpoints from a) to d):
Run check in Backdoor Incident Response Matrix for node(v_i).

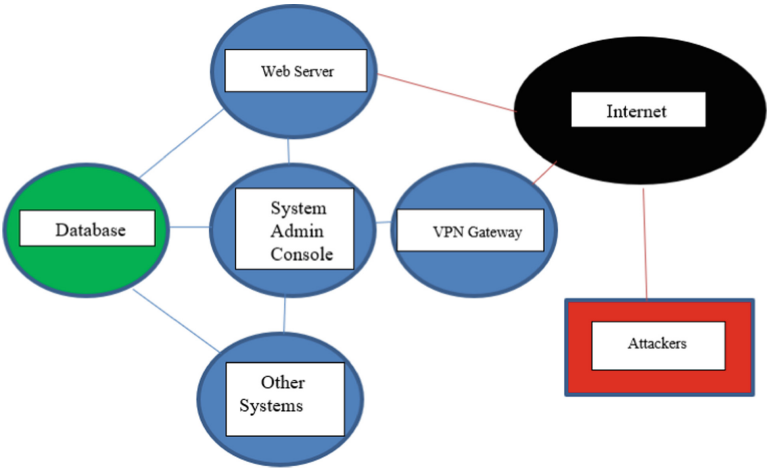

Fig. 4. High level graph of node deployment in online gaming company.

For the matrix, it covers backdoor, backdoored server and network traffic illustrated with the following table. We have demonstrated the following methodology in our real-world backdoor detection in an online gaming company. We have added several analyses which may not be applicable in our incident, however, it will be useful in other types of backdoor incidents for generic analysis purposes (Table 1).

3.1 Static Analysis of Backdoor

We have investigated into the backdoor sample Transtatic.dll (SHA256: 124fd83e874b36dafbc87903037e4c014d81e699b523339ce46e93d4dab772da) from an online gaming company, successfully identified the following characteristics:

1. They support upload, download, and command execution functions. The Backdoor manipulates a stealthier approach by taking Content-Type in HTTP header [20] with "IMAGE/type" as png, jpg, and gif, so as to upload data to, download data from, and executing a command against the victim IIS Web server (Fig. 5a).
2. Response data is sent with the Content Type "Text/Plain".
3. When both Malware upload the data, they will detect the end of upload completion if an error or failure message is found after a while loop of data transfer (Fig. 5b). The backdoor will use GetTickCount function to count the time when uploading the data, which is also aligned with the Malware writer's preference in calculating the time (Fig. 5c).
4. Keywords and API used in both Backdoor samples include *RegisterModule, CHttpModule, "Server Error", "Failed",* etc.

In view of the above findings, we have considered these IIS native module backdoor indicators and developed a scanner, thereby giving different indicators to server administrator for reference, whether he/she can carry out further investigation over suspicious Microsoft Web Server IIS native modules.

Table 1. BackDoor Incident Response Matrix (BDIRM)

1. Backdoor	2. Backdoored Server	3. Network Traffic
1.1 a. Identify the function capability of the suspicious backdoor [11], which is helpful to reverse engineering. b. Carry out static analysis and reverse engineering of the backdoor binary or/and webshell. It helps to update search rules in 3.3. c. Emulate the binary in emulation framework [12] for dynamic analysis. d. Write Yara rules [13] with artifacts found from items a – c. It will be useful to scan and discover any backdoors in other nodes of machines or systems.	2.1 Prepare the golden copy of configuration and deployment files.	3.1 Benchmark the common request/response header. Prerequisite: SIEM [14] or proper logging is deployed.
1.2 Carry out similarity check with published backdoor whether they share similar code instruction structure and commands with binary differing [15] and TLSH[16]. It helps to update search rules in 3.3.	2.2 Compare the golden copy of configuration and deployment files with those with the potentially hacked server.	3.2 Monitor high volume of HTTP request/response. Monitor unpopular incoming and outgoing IP addresses.. Prerequisite: SIEM or proper logging is deployed

(*continued*)

Table 1. (*continued*)

1.3 Scan all DLLs to detect any abnormally high number of encoded base64 strings, obfuscated SQL and Powershell scripts and possible system command(s) and file upload/download/delete command(s).	2.3 Examine activity and error logs of the hacked server to detect any successful or failed loading of suspicious external programs or/and modules.	3.3 Write search rules to detect and alert unpopular user agent and cookie values from the logs. Write search rules to detect and alert suspicious requests based on findings in 1.1, 1.2, 1.3 and 2.4.
1.4 Apply Shannon entropy detection over the binary to detect obfuscated, compressed and/or encrypted code distribution [17].	2.4 Export and dump the volatile memory of the backdoored server. Repeat 1.3, scan and detect any abnormally high number of encoded base64 strings, obfuscated SQL and Powershell scripts and possible system command(s) and file upload/download/delete command(s). [18]	3.4 Network traffic deobfuscation: Detect encrypted and encoded traffic. [19]

```
v9 = *(_BYTE **)(v7 + 8);
if ( *v9 != 'I' || v9[1] != 'M' || v9[2] != 'A' || v9[3] != 'G' || v9[4] != 'E' || v9[5] != '/'
  return 0i64;
v10 = v9[6];
if ( v10 == 'j' )
{
  if ( v9[7] == 'p' && v9[8] == 'g' )
  {
    sub_180002360(v9, (unsigned int)v8, &v34);
```

Fig. 5a. Stealthy approach of data upload, data download and command execution in Transtatic.dll. It takes in IMAGE/jpg as part of Backdoor command to download victim data from backdoored Microsoft IIS Web server.

```
v3,
500i64,
"Server Error",
0i64,
```

Fig. 5b. Provide "Server Error" message when data is downloaded in Transactic.dll.

```
LABEL_47:
    if ( v19 != v20 )
    {
      v23 = v19 + 8;
      do
      {
        if ( GetTickCount() - *(_DWORD *)(*(_QWORD *)v19 + 80i64) < 0xEA60 )
```

Fig. 5c. Use of GetTickCount in upload function. Attacker enters IMAGE/png as command to upload the data to the backdoored IIS Web server.

We realize there is research about detecting trigger-based Malware based on network traffic and machine learning approaches; the assumption is we can identify the backdoor traffic and have samples for supervised learning. However, the samples in the IIS native module are missing, and it will cause a high degree of false positives and negatives.

3.2 Similarity Comparison with published Backdoor

According to our matrix, we carry out a binary similarity comparison between the OilRig and our revealed backdoor and see whether they come from the same APT campaign or family. To this end, we use a popular binary code diffing tool named BinDiff [21]. We report the overall similarity score is 0.18 (Fig. 6a) and for the function responsible for uploading and downloading data, the similarity score is 0.27 (Fig. 6b). The similarity is useful and applied to identify variants (Fig. 6c) instead of a completely different purpose IIS native module backdoor.

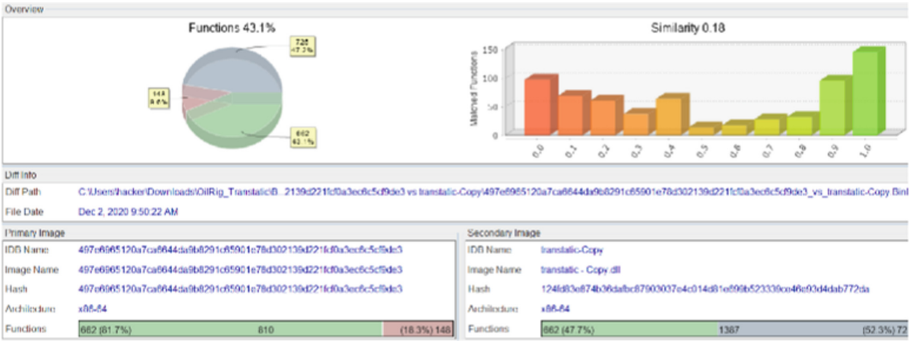

Fig. 6a. Binary similarity between OilRig and online transtatic backdoors.

Similarity	Confidence	Address	Primary Name	Type	Address	Secondary Name
0.27	0.51	0000...	sub_180002580	Normal	0000...	sub_180003620
0.26	0.27	0000...	sub_180001FA0	Normal	0000...	sub_180003020

Fig. 6b. Function similarity. It indicates the upload/download function similarity as 0.27 between OilRig and transtatic backdoors.

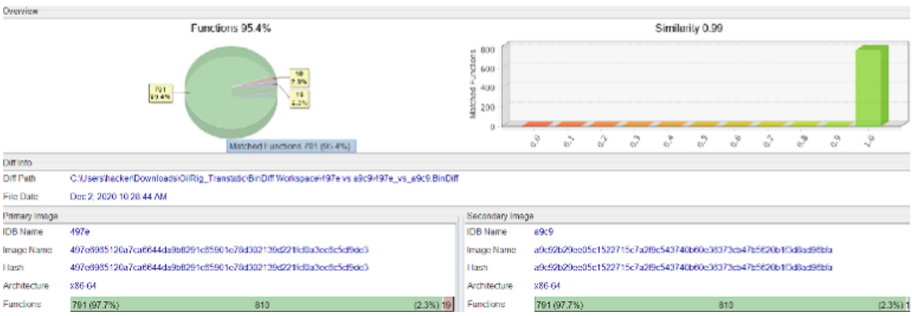

Fig. 6c. Variant similarity. It exhibits similarity value as 0.99 for OilRig backdoor variants. There are 19 functions that are modified.

3.3 Scanning Potential Backdoors

The scanner has the following functions for string-based detection and assembly instruction-based detection based on our investigation and correlation of our findings:

String-Based Detection

Other than the keyword and string pattern matching, we have attempted to identify whether the SQL strings and Powershell scripts are obfuscated through cosine similarity comparison between vectors of collected strings from SQL/Powershell script repositories and the target native module DLL file. The Cosine similarity is used in detecting obfuscated code in viruses [22] to detect virus variants more effectively. The more obfuscated strings of SQL and Powershell scripts are found, the higher the probability it is a backdoor Malware. Here is our methodology:

1. Build the frequency table of characters over files. By using the tool, we generate two standard Top 30 Character Frequency tables (Fig. 7a and Fig. 7b) and attached full tables in Appendix, one for SQL files and the other one for powershell files. The legitimate SQL and Powershell scripts files are collected from Microsoft's [23] and Azure's [24] GitHub repositories, respectively.

2. Extract the strings from a target file by using "Strings". It will scan a file for Unicode or Ascii strings of a default length of 3 or more characters.
3. Classify the strings by finding the language keywords. If the string contains SQL language keywords, it will be classified as SQL language. If it also contains PowerShell language keywords, it will also be classified as PowerShell language.
4. Build the frequency table of the classified strings.
5. Calculate the cosine similarity of the frequency table of the classified strings and the frequency table of the standard language files (Fig. 7c).
6. Find all base64 encoded strings.
7. Find the specific keywords, such as *"DownloadString", "upload", "download", "chttpmodule", "encode", "decode"*.

Character	Frequency
0	0.087421
<Space >	0.075127
e	0.047623
a	0.034788
3	0.034726
6	0.034533
C	0.033376
4	0.033155
,	0.032057
9	0.032014
t	0.031492
1	0.025061
i	0.024927
r	0.022208
o	0.021458
n	0.021324
5	0.019975
'	0.019325
c	0.019143
d	0.017487
D	0.017420
7	0.016492
2	0.016190
F	0.015992
E	0.013884
8	0.013748
m	0.012664
l	0.010761
T	0.010660
S	0.010444

Fig. 7a. Top-30 Character Frequency table for SQL file.

Character	Frequency
<Space>	0.168742
e	0.085947
t	0.056438
r	0.055300
o	0.052609
a	0.048058
i	0.038527
s	0.036009
n	0.035964
c	0.030310
-	0.025493
u	0.025067
m	0.020010
p	0.019976
l	0.019949
$	0.018316
d	0.016633
g	0.013452
"	0.012247
N	0.011838
S	0.010192
A	0.010040
h	0.009941
.	0.008611
y	0.007967
b	0.007843
f	0.007659
I	0.007238
P	0.006948
R	0.006778

Fig. 7b. Top-30 Character Frequency table for Powershell files.

Experiments

We have scanned our current backdoor samples and legitimate Microsoft IIS native module DLL files under the Microsoft Web Server IIS system folder (Fig. 7c).

```
C:\Users\hacker\Downloads\iis_backdoor_scanner-main\iis_backdoor_scanner-main>python38 program.py

Target file:  C:\Users\hacker\Downloads\iis_backdoor_scanner-main\iis_backdoor_scanner-main\sample\2020-04-transtatic.dl
l
Total string:  13824
Number of sql string:  148
Number of ps1 string:  666
Number of specific keyword string:  5
Number of base64 string:  33

Similarity of sql 0.6792574371217315
Similarity of ps1 0.8731818119026385
Find special functions:  [{'string': ' but did not override the method in its CHttpModule implementation.  Please check
the method signature to make sure it matches the corresponding method.', 'match': ['CHTTPMODULE']}, {'string': 'text/pla
in', 'match': ['TEXT/PLAIN']}, {'string': 'GetTickCount64', 'match': ['GETTICKCOUNT64']}, {'string': 'RegisterModule', '
match': ['REGISTERMODULE']}, {'string': 'GetTickCount', 'match': ['GETTICKCOUNT']}]
```

Fig. 7c. Scanning Statistics of Backdoor.

From Table 2, we can find both backdoor malwares hit a high number of keyword and API indicators, SQL strings and Powershell script strings. The obfuscation index on both SQL and Powershell scripts do not exhibit significant deviation between malware backdoor and the legitimate native module. However, standard Microsoft native modules, CustomError and HTTPCache, have been found to have indicators similar to backdoor behaviors. This requires further investigation over binary signatures and hash to determine whether an existing library is backdoored or not.

Table 2. String-based Backdoor Indicators.

Native module filename	No. of SQL strings	No. of ps1 strings	No. of interesting strings/keywords and API	No. of Base64 encoded strings	Obfuscation index SQL	Obfuscation index ps1
Backdoor: Transtatic.dll from Online Gaming incident (SHA256 124fd83e874b36dafbc87903037e4c014d81e699b523339ce46e93d4dab772da)	148	666	5	33	0.679	0.873
OilRig (Backdoor) (SHA256 497e6965120a7ca6644da9b8291c65901e78d302139d221fcf0a3ec6c5cf9de3)	82	161	32	12	0.627	0.938
AnonymousAuthenticationModule	5	19	3	0	0.649	0.931
CustomErrorModule	2	52	3	4	0.625	0.597
DefaultDocumentModule	5	9	3	1	0.625	0.789
DirectoryListingModule	4	13	3	2	0.620	0.730
HTTP Cache Module	5	48	3	2	0.627	0.599
HTTPLoggingModule	4	34	3	2	0.645	0.805
ProtocolSupportModule	4	11	3	1	0.624	0.804
RequesteFilteringModule	21	26	4	0	0.650	0.863
StaticCompressionModule	2	26	3	5	0.624	0.730
StaticFileModule	8	29	3	4	0.628	0.749

Assembly Instruction-Based Detection
Previously, we have observed the outbound data is exported via "text/plain" with command strings only. In our scanner, we suggested the following instruction-based detection methodology to provide a backdoor indicator, as the backdoor needs a command to trigger every action and upload and download the target victim information at the Web server:

- Search "text/plain" and other content-type values in the target IIS native module DLL file.
- Locate the function that contains the reference to the "text/plain" or other content-type values string.
- Identify all *cmp* instructions and *strstr* instructions that are used for comparison with operands with ASCII character code only. For Windows binaries, we will identify invocations of *CompareStringA* function.
- If high numbers of *cmp, strstr* and *CompareStringA* are found under a data export function, we deem this as an indicator of a backdoor.

Table 3. Instruction-based backdoor indicators.

Native module filename	No. of CMP	No. of StrStr	No. of calling CompareStringA	Identified character sequence and command
Transtatic.dll (Backdoor)	25	0	0	Yes
OilRig (Backdoor)	32	4	0	Yes
IIS RAID (Backdoor)	0	0	4	Yes
AnonymousAuthenticationModule	0	0	0	No
CustomErrorModule	0	0	0	No
DefaultDocumentModule	0	0	0	No
DirectoryListingModule	0	0	0	No
HTTPCacheModule	0	0	0	No
HTTPLoggingModule	0	0	0	No
ProtocolSupportModule	0	0	0	No
RequestFilteringModule	0	0	0	No
StaticCompressionModule	0	0	0	No
StaticFileModule	0	0	0	No

We have included a summary table of indicators (Table 3) for both malicious and legitimate native modules. We have found that numbers for malicious backdoor with higher numbers in *CMP*, *StrStr* and *CompareStringA* compared with other legitimate native modules, and it satisfies our assumption: a typical native module to extend Web

server capability will not engage command execution and SQL queries into business data in a database.

3.4 Deobfuscation of Binary and Network Traffic Content

Capturing network traffic of a potential compromised host is crucial for further analysis. Attackers prefer to use various encoding including ROT-13 encoding, URL encoding, simple XOR encryption techniques and string obfuscation techniques to hide their commands and activities [25].

In the incident response scenario, the database and Web server transaction, we propose the following algorithm to capture and analyze the traffic on incremental time snapshot basis, which is practically reduce the performance overheads caused by network traffic capture:

Algorithm 2. Detection of obfuscated data in captured network traffic on incremental

For every suspicious node (v_i) (with which the server connects to untrusted parties):

a. Identify all timestamps (T_i) of compromised transactions and change of files.
b. For each previous incident timestamp T_i, we start monitoring at T_i +/- j where j= 1, 5,10,15,30, 45 and 60 minutes at node (v_i):
   Capture the network traffic (N_{ij}).

For every captured network traffic (N_{ij}):

a. Apply XOR string detection.
b. Apply URLencode and ROT-13 encoding detection.
c. Apply obfuscated string detection.

Detect if any of (a - c) is satisfied.

3.5 Backdoor Entropy Analysis

Attackers need to pack the binaries or/and any files which are helpful to their attack in a stealthy manner, including software packing, compression and encryption. In our mentioned incident, all the artifacts are presented in clear text, the string-based and pattern matching techniques are good enough to identify. However, the scope of examination is incomplete. Here is the algorithm to detect any suspicious backdoor or obfuscated/encrypted/compressed files in the node with Shannon Entropy [17].

Algorithm 3. Detection of obfuscated files in node with Shannon Entropy

Entropy ← Array[][]

For every node v_i:
  For each server configuration c_j:
    Entropy[i][j] = Shannon entropy(v_i, c_j)

For every Entropy[i][j] of calculated Shannon entropy:
  a. Compare with the freshly built configuration file(s) and system file(s).
  b. Identify all discrepancy of entropy value and carry out further examination of the suspicious file(s).
  c. Identify all high entropy values and carry out further examination of the suspicious file(s).

4 Related Work

In backdoor detection-related research areas, there are no similar publications to innovate an incident response model for Backdoor investigation, however, there are still individual research approaches to Backdoor analysis researchers suggest detecting interactive network traffic streams in non-standard service port [26]. There is research about detecting malware via threshold random walk model in port scanning, however, this approach can be evaded if an attacker has ability to create high-quality hit lists with well-designed active intervals [27].

Attack graph [28] is a representation for all possible paths of attack towards a network or system. It's a widely used technique on identifying the most critical regions of a system toward attack scenarios. For example in [29] the author has used Minimum Cost SAT solver (MCSS) and iteratively distilled critical attack graph surface from a full attack graph. In their paper, they suggested that the attacker will pick the attacking route with minimum effort. Hence, MCSS can be used to locate the minimum-cost attacking route in a network. In [30] the author proposed an approach to measure the likelihood of lateral movement happening on a selected node by considering the reachability of it from any arbitrary nodes in the attack graph. A graph-based system called Latte [31] to identify malicious lateral movement path and the network connection graph is established with Kerberos service ticket requests and another research reveals lateral movement using authentication logs [32], which symbolises the criticality of authentication logs management.

Log2vec [33] is a deep learning approach that aims at detecting APT attacks or user's malicious behavior through capturing and representing rich relationships among user's operations. However, this approach assumed victim user's account will perform various activities that prepare for lateral movement, which may not be the case in a lot of APT attack.

5 Discussion

Backdoor investigation is a challenging task because of its stealthy and trigger-based nature. Once we have identified the incident and the impacted server, we are required to trace and figure out how the attacker gets into the system. Network and systems logs may not be adequate for investigation especially when parameters used by attackers to deliver the payload and commands are not logged.

We have suggested a Backdoor IR matrix, there are limitations in different scenarios. In binary emulation, the emulation can be successfully done if we can develop all necessary API calls in the framework, otherwise, the emulation will stop because of missing several API libraries. Suspicious backdoor binaries and modules must be loaded and emulated in another server with the same version and patch level for further analysis instead of taking a binary emulation approach.

Outsourcing to operating system product vendors for investigation and forensic analysis is challenging. The vendor has requested the online gaming customer to access the production database remotely for forensic study, which is not in compliance with enterprise cybersecurity policy. The vendor has provided forensic tools and requested the customer to install and execute, however, the memory image dump causes the performance and file system memory overhead, which are not feasible in our incident response.

As an incident handler, business owners are always concerned about whether there is no more malware installed in other systems, other than figuring out the impacted systems and compromised servers along with the attack path, we still need to scan and find out any other backdoor and malware deployed by attackers. In consideration of business impact, the business owner is reluctant to take down the system and prefer live incident response.

6 Conclusion

In this paper, we propose a BackDoor Incident Response Model (BDIRM) to investigate backdoor incidents, analyze backdoor, backdoored server and network traffic with application. The proposed model has been successfully implemented with our online gaming customer, identified and eradicated the incident. On top of the existing industry incident response guidelines, we contribute this systematic and practical model to the incident response community to identify and detect highly stealthy backdoor malware.

Acknowledgement. I would like to express my very great appreciation to my supervisor, Dr. Jogesh Muppala, who gives detailed guidance to my research and valuable advice and review from Dr. Shuai Wang. Moreover, I am very thankful to my co-authors including Ken Wong, Austin Lau and Johnny Wong who have contributed to the model design, carried out experiments and tools development and Alan Ho as we carry out deep reverse engineering over the backdoor. I appreciate Dr. Zetta Ke and Dr. Dongsun Kim to review my paper and Halvar Flake and Christian Blichmann to share and discuss deep and modern Malware and Backdoor detection related research with me. Finally, I thank anonymous reviewers for their valuable feedback. This work was supported in part by a Bridge Gap Fund of TTC/HKUST (Project ID: BGF.003.2021).

References

1. Falcone, R.: OilRig uses RGDoor IIS Backdoor on Targets in the Middle East (2018). https://unit42.paloaltonetworks.com/unit42-oilrig-uses-rgdoor-iis-backdoor-targets-middle-east/. Accessed 7 December 2020
2. Baker, P.: The SolarWinds hack timeline: Who knew what, and when? (2021). https://www.csoonline.com/article/3613571/the-solarwinds-hack-timeline-who-knew-what-and-when.html
3. Cichonski, P., Millar, T., Grance, T., Scarfone, K.: Computer Security Incident Handling Guide. CSRC (2012). https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final
4. Incident Response SANS: The 6 Steps in Depth. Cynet. (n.d.). https://www.cynet.com/incident-response/incident-response-sans-the-6-steps-in-depth/
5. Liu, K., Tang, S., Wan, J., Ding, S., Qin, X.: Improved layed architecture for developing semantic web application. In: 2010 2nd International Conference on Future Computer and Communication, vol. 3, pp. V3–769. IEEE (2010
6. VirusTotal, Virustotal.com (2020). https://www.virustotal.com/gui/
7. Li, F., Lai, A., Ddl, D.: Evidence of advanced persistent threat a case study of malware for political espionage. In: 2011 6th International Conference on Malicious and Unwanted Software, pp. 102–109. IEEE (2011)
8. Hardy, S., et al.: Targeted threat index Characterizing and quantifying politically-motivated targeted malware. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14, pp. 527–541 (2014)
9. CSIRT Services Framework Version 2.1. FIRST (2019). https://www.first.org/standards/frameworks/csirts/csirt_services_framework_v2.1
10. Command and Control. Command and Control, Tactic TA0011 - Enterprise | MITRE ATT&CK® (2018). https://attack.mitre.org/tactics/TA0011/
11. Fireeye. fireeye/capa. GitHub (2020). https://github.com/fireeye/capa
12. Qilingframework. qilingframework/qiling. GitHub (2020). https://github.com/qilingframework/qiling
13. VirusTotal. VirusTotal/yara. GitHub (2013). https://github.com/VirusTotal/yara
14. Detken, K., Rix, T., Kleiner, C., Hellmann, B., Renners, L.: SIEM approach for a higher level of IT security in enterprise networks. In: 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), pp. 322–327 (2015). https://doi.org/10.1109/IDAACS.2015.7340752
15. Haq, I.U., Caballero, J.: A survey of binary code similarity. ACM Comput. Surv. **54**, 3, Article 51, 38 (2021). https://doi.org/10.1145/3446371
16. Trendmicro. trendmicro/tlsh. GitHub (2015). https://github.com/trendmicro/tlsh
17. Lyda, R., Hamrock, J.: Using entropy analysis to find encrypted and packed malware. IEEE Secur. Privacy Magaz. **5**(2), 40–45 (2007). https://doi.org/10.1109/msp.2007.48
18. https://www.fireeye.com/blog/threat-research/2016/06/automatically-extracting-obfuscated-strings.html
19. https://resources.infosecinstitute.com/topic/network-traffic-analysis-for-ir-content-deobfuscation/
20. MIME types (IANA media types) (2019). https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. Accessed 7 December 2020
21. zynamics.com - BinDiff (2020). https://www.zynamics.com/bindiff.html. Accessed 7 December 2020
22. Karnik, A., Goswami, S., Guha, R.: Detecting obfuscated viruses using cosine similarity analysis. In: First Asia International Conference on Modelling and Simulation (AMS 2007), pp. 165–170. IEEE (2007

23. microsoft/sql-server-samples (2020). https://github.com/microsoft/sql-server-samples. Accessed 7 December 2020
24. Azure/azure-docs-powershell-samples (2020). https://github.com/Azure/azure-docs-powershell-samples. Accessed 7 December 2020
25. Cannell, J., and ABOUT THE AUTHOR Joshua Cannell Malware Intelligence Analyst.. Obfuscation: Malware's best friend. Malwarebytes Labs (2016). https://blog.malwarebytes.com/threat-analysis/2013/03/obfuscation-malwares-best-friend/
26. Zhang, Y., Paxson, V.: Detecting backdoors. In Proceedings of the 9th conference on USENIX Security Symposium - Volume 9 (SSYM 2000), vol. 12. USENIX Association, USA (2000).
27. Jung, J.: Real-Time detection of malicious network activity using stochastic models. Ph.D. thesis, Massachusetts Institute of Technology (2006)
28. Jha, S., Sheyner, O., Wing, J.: Two formal analyses of attack graphs. In: Proceedings of the 2002 Computer Security Foundations Workshop, pp. 45–59, Nova Scotia (2002)
29. Huang, H., Zhang, S., Ou, X., Prakash, A., Sakallah, K.: Distilling critical attack graph surface iteratively through minimum-cost sat solving. In: Proceedings of the 27th Annual Computer Security Applications Conference. ACM, pp. 31–40 (2011)
30. Johnson, J.R., Hogan, E.A.: A graph analytic metric for mitigating advanced persistent threat. In: 2013 IEEE International Conference on Intelligence and Security Informatics (ISI). IEEE, pp. 129–133 (2013)
31. Liu, Q., et al.: Latte: Large-Scale Lateral Movement Detection. MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM), pp. 1–6 (2018) https://doi.org/10.1109/MILCOM.2018.8599748
32. Bian, H., Bai, T., Salahuddin, M.A., Limam, N., Daya, A.A., Boutaba, R.: Uncovering lateral movement using authentication logs. IEEE Trans. Netw. Serv. Manage. **18**(1), 1049–1063 (2021). https://doi.org/10.1109/TNSM.2021.3054356
33. Liu,, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., Meng, D.: Log2vec: a heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In: Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1777–1794 (2019)