Feature

# Persistent Cross-interface Attacks

**Aditya K. Sood** is a security researcher and doctoral candidate at Michigan State University (USA) and has worked in the security domain for Armorize, COSEINC and KPMG. Sood is a founder of SecNiche Security Labs, an independent security research arena for cutting-edge computer security research.

**Richard J. Enbody, Ph.D.,** is associate professor in the Department of Computer Science and Engineering at Michigan State University (USA), where he joined the faculty in 1987. His research interests include computer security, computer architecture, web-based distance education, and parallel processing. Enbody has two patents pending on hardware buffer-overflow protection that could prevent most computer worms and viruses.



With the advent of new technologies, exploitation trends have shifted from systems to the web. This is due to the fact that system vulnerabilities are getting harder to exploit, while exploitation through the web is made easy via infecting web sites with malware. A number of network devices are using web and back-end interfaces for administrative operations. These network devices include disk stations, firewalls, routers, modems and switches that are used heavily in home and organizational networks. The back-end administration interfaces include File Transfer Protocol (FTP) and Telnet. No doubt, the web has provided ease and flexibility, but it has greatly impacted the privacy and security of users and is prone to malware infections. Web interfaces also provide ease and flexibility, but at a cost: These web-facing network devices are prone to security vulnerabilities.

This article explores the types of attacks and vulnerabilities that persist within network devices supporting both web and traditional administrative interfaces. The vulnerability of one network-based storage device, Synology DiskStation Manager [DSM], was selected for analysis. The goal is to understand the vulnerability and exploitation technique used to conduct a successful cross-interface attack (CIA), which is defined as the use of one interface to attack another interface. For example, an FTP console interface can be used to attack a web interface. Knowledge and understanding of these types of attack patterns are required to conduct aggressive testing while performing security assessments.

To conduct any specific category of attacks, both an appropriate entry point and an attack surface are required. Efficient attacks result in gaining maximum information with minimum intervention in the system.[1] There are a number of cross-site request forgery (CSRF) vulnerabilities that have occurred in network devices,[2, 3, 4] but most of them follow a similar pattern, requiring exploitation of browser inefficiencies.[5] CSRF is a class of attacks in which the attacker exploits the user session to execute commands on the running server or in the application itself.[6] These attacks are prevalent in web applications and require robust solutions;[7, 8] however, it has been noticed that web applications used for managing network devices are usually designed in an insecure way. A related type of attack has targeted the Juniper firewall.[9] Attacks through back-end consoles have not been explored much.

The vulnerability discussed in this article was detected in Synology DSM and was disclosed to the vendor,[10, 11, 12] and the vulnerability is patched in the latest versions of Synology DSM. Using a step-by-step approach, this article explores the attacking technique related to this vulnerability and how back-end login (administrative) consoles are used to launch CIAs. Remote command execution through CSRF is quite reliable in this sort of attack, and certain techniques follow a tactical way of enumerating the information from the consoles that can also be helpful in launching attacks in vulnerable applications.[13] If the network device follows the vulnerable design pattern explained in this article, an attacker can perform CIAs.

## THREAT MODEL

An attacker can exploit the remote command execution vulnerability by injecting commands through the FTP administrative console (i.e., username and password) to render payloads in the web application interface of network devices. There are a variety of injections that an attacker can use to exploit this type of vulnerability. The attacker's goal is to exploit the design of back-end login consoles (FTP consoles). Remote code execution through CSRF includes the possibility of adding users, deleting records and modifying configurations on the network device management module per the attacker's need.

The threats associated with this vulnerability are enumerated based on the risks as follows:

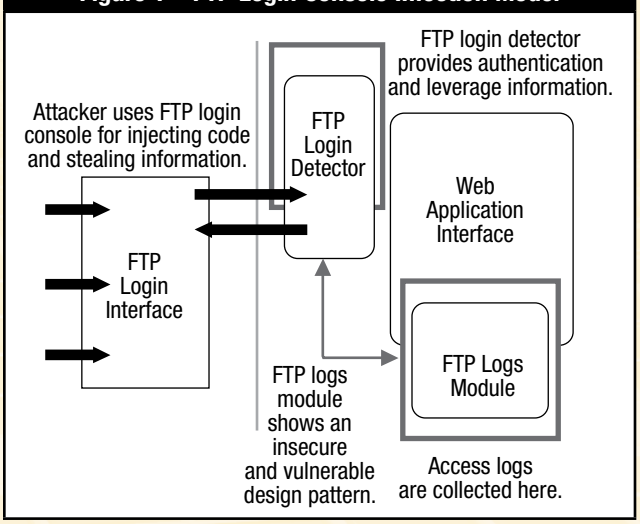
- **Remote command execution through CSRF**—This type of vulnerability addresses the remote code execution behavior through the CSRF strings that are injected through the input point and are rendered in the web application to produce the desired impact. The possibility of remote code execution is higher because administrative privileges are used to conduct these types of attacks in the system.
- **Malware infections**—Cross-site scripting vulnerabilities allow the attacker to inject malicious links in the web interface. Upon executing such a link, the malware is automatically downloaded into the server. The attacker can inject code to execute drive-by download attacks or further exploit the vulnerabilities of the running web server to increase the infection rate.[14, 15] iFrame injections lead to a more subtle malware infection. The Hypertext Transmission Protocol (HTTP) specification allows the effective use of iFrame to embed one web page into another web page. The embedding is irrespective of the domain to which a page belongs, and can be used in a cross-domain context. Since iFrames are interactive in nature, it is possible to bypass a same origin policy (SOP), and it is easy to launch cross-domain attacks (CDA) if a certain set of vulnerabilities exists in the base software or in web applications.
- **Information theft**—In broad terms, the information regarding administrative sessions can be compromised through cookie-stealing attacks. The injected code leverages the already-set-up session and transfers the cookie to a third party, thereby resulting in theft of the information from the system.

## UNDERSTANDING THE DESIGN AND VULNERABILITY

Most network-facing devices provide multiple ways to access firmware, such as by having the administrative web interface running on a specific port or back-end access through FTP or Telnet consoles. A similar set of vulnerabilities has been observed across network devices that can be exploited in a generic manner. For example, misconfiguration can result in compromise of the network device through administrative consoles that are used for login purposes.

For testing and conducting attacks, it is useful to understand the design of the application. Automated testing to find patterns provides an edge in determining the loopholes and workflow of the deployed system. Manual testing can expose hidden functionality of web applications that cannot be found using automated tools alone. Generally, the FTP login console provides an FTP logging (error) mechanism. This means that any unsuccessful attempt to connect to the FTP server is logged in the web application logging module. Network device firmware uses the web application interface for all types of operations. Any user who logs into the application through the FTP console will be scrutinized through the FTP login detector module, and the response is sent back to the user (or attacker). Meanwhile, the credentials provided by the user are logged in the FTP logs module; the details are presented in **figure 1**.

**Figure 1—FTP Login Console Infection Model**




The vulnerability and inconsistency are seen in the fact that errors generated at the FTP login console are stored in the access logs in the web application. Many devices do not use appropriate filtering mechanisms; hence, injected payload is executed in the context of web application, resulting in remote code execution through, for example, CSRF or malware infection. The vulnerability revolves around the FTP logging mechanism. Primarily, the FTP logs are rendered as text or HTML content in the web interface, depending on the content-type parameters. Whatever input is supplied by the attacker is inserted as a log entry. An important observation is that the log process runs as "administrator" or "root," which means that any malicious code injected into logs will run in privileged mode. Further, the logs are not rendered in a customized format, which allows Document Object Model (DOM) and HTML elements to be rendered as dynamic content in the browser's JavaScript interpreter. In addition, no specific security mechanism is

applied, such as a generic sandbox, to avoid the code execution through injected parameters. Of course, in this context, proper security would filter input prior to logging it. This is a design problem that has been found in many network devices.[16]

## ATTACK STEPS AND TECHNIQUES

This section details a single type of attack, but many variations can work:

1. Validating the default buffer length in the FTP console
2. Fingerprinting FTP error codes
3. Injecting payloads

### Step 1—Validating the Default Buffer Length in the FTP Console

In the first step, the attacker tries to validate the default buffer length accepted by the FTP console. The attacker can tactically enumerate information available from administrative consoles, which is useful in building an attack. The attacker can determine the possible length of the string of a username that is accepted by the FTP server. To get this information, the attacker can use a default buffer trick, as presented in **figure 2**.

The 331 response code shown in **figure 2** provides information about the username and asks for the password. The attacker can easily compute the string length that is acceptable in the username field. In this case, it is approximately 80–100 characters. Once this information is known, it becomes easier for the attacker to design reliable injections pertaining to the limit of accepted characters. This information can be exploited through back-end login consoles.

**Figure 2—Determining the Acceptable Buffer Length**

Analyzing String through Default Buffer Trick

```
root@redux$ ftp example.com
Connected to example.com.
220 Disk Station FTP server at DiskStation ready.
User (example.com:(none)):
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
331 Password required for
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA.
Password:
530 Login incorrect.
Login failed.
```

### Step 2—Fingerprinting FTP Error Codes

In the second step, the attacker fingerprints the error code returned by the running FTP server and any input that is passed through the FTP console. It is possible to use the FTP server consoles to inject payloads into the username and password fields. In this way, an attacker can use a back-end login console to inject unauthorized code. The default design of FTP allows for the acceptance of both the username and password prior to the authentication process; however, there are a number of FTP servers that perform user verification prior to authentication. User verification without a password in the authentication process is not a good security practice because the attacker can enumerate users on the system by interpreting the errors. It indirectly helps in enumerating the users, as presented in **figure 3**.

**Figure 3—Enumerating User Accounts Based on FTP Error Codes**

Customized PERL Script Verifying Users

```
$ perl ftp_user_reconnaisance.pl example.com
(*) written by- adi_ks [at] secniche.org

(*) anonymous access allowed - example.com
(*) example.com does not support TLS
(*) FTP username verification
(*) enumerating system accounts

conn str 0] [temp] does not exist
conn str 1] [root] exists
conn str 2] [bin] exists
conn str 3] [daemon] exists
conn str 4] [adm] exists
conn str 5] [lp] exists
conn str 6] [sync] exists
conn str 7] [shutdown] exists
conn str 8] [halt] exists
conn str 9] [mail] exists
conn str 10] [news] exists
conn str 11] [uucp] exists
conn str 12] [gopher] does not exist
conn str 13] [apache] does not exist
conn str 14] [named] does not exist
conn str 15] [amanda] does not exist
conn str 16] [indent] does not exist
conn str 17] [rpc] does not exist
.....

(*) command completed successfully.
```

In this example, the normal functionality of an FTP server is manipulated by the attacker, as arbitrary code is injected as input into the username and password prompts. Of course, the authentication fails, but it generates connection failure errors in the web application logs. The error-handling mechanism used in the FTP console provides information, such as the presence of user accounts in the system, to the attacker that is quite useful in generating an attack surface. For example, a malicious input injected through an FTP login console shows the acceptable string length, thereby prompting for a password, such as the 331 response code discussed earlier. The error-reporting mechanism should be used in conjunction with the FTP authentication module to restrict the acceptance of malicious input through login consoles. FTP generates errors, as presented in **figure 4**.

**Figure 4—FTP Error Messages That Are Programmed to Enumerate Users**

| No. | FTP Error Messages |
|---|---|
| 1 | Login incorrect |
| 2 | Login with USER first |
| 3 | 503 You are already loggin in! |
| 4 | Permission denied |
| 5 | Login authentication failed |

### Step 3—Injecting Payloads

In the final step, the attacker injects a payload in the acceptable buffer length. The details of this step are explained in the case study discussed in the next section. In the design of some web applications, security weaknesses exist in the form of dependencies among different modules, allowing access to the underlying system. Complex designs result in security failures. This defect has shown the stature of insecure and inappropriate design of web interfaces in network devices.

## CASE STUDY—SYNOLOGY DISKSTATION MANAGER

This section discusses a specific vulnerability in Synology DSM to illustrate how an attacker can exploit the environment. Synology DSM is a network device used by organizations to manage data in a centralized environment. Of course, it is a hardware-based solution with administrative interfaces. Before examining the vulnerability pattern, it is advised to determine the application design and a generalized flow of information.

The vulnerability in Synology DSM was discovered as a result of the remote compromise of the product due to a weak configuration that provided access to all modules with highest privileges. It provided an inbound and outbound control that could allow an attacker to scrutinize the way in which the product handled the input code from different interfaces. The product uses GET requests to manage the control of logging activity, as presented in **figure 5**.

**Figure 5—Request Mechanism in Synology DSM**

```
GET /webman/modules/logman.cgi dc=1273595767787 &action=
view&start=0&limit=50&logtype=connlog &sort=time&dir=DESC
HTTP/1.1

GET /webman/modules/logman.cgi dc=1273595786011 &action
=view&start=0&limit=50&logtype=syslog &sort=time&dir=DESC
HTTP/1.1
```

The user is added through userman.cgi, as presented in **figure 6**.

**Figure 6—Request/Response Mechanism for Adding a User**


Synology DSM uses a normal POST request for adding a user by submitting a form with variables. On further analysis of the application, it was detected that the product allows the FTP login console as an entry point to inject data. By

default, the FTP login console does not scrutinize the types of strings that are passed. The attacker can inject malicious code or attack payloads because strings are not filtered. The injected rogue code is not a registered FTP user; as a result, the login is not allowed and fails with an error. Consequentially, it is noticed that errors are logged in the FTP connection log module with inappropriate content-type specification, which allows the injected code from the FTP login console to render code as HTML, JavaScript, etc. The injection occurs in the context of admin privileges, which increases the impact of the attack.

To illustrate the attack, a differential set of tests was performed, as presented in **figure 7**.

**Figure 7—Code Injected as Inputs Through the Login Console**

```
root@redux$ ftp example.com
Connected to example.
220 Disk Station FTP server at DiskStation ready.
User (example.com:(none)):
" >< imgsrc =' Z' / >
331 Password required for
" " >< imgsrc =' Z' / >
Password:
530 Login incorrect. Login failed.
```

```
root@redux$ ftp example.com
Connected to example.
220 Disk Station FTP server at DiskStation ready.
User (example.com:(none)):
" >< scriptsrc =' Z' / >
331 Password required for
" " >< scriptsrc =' Z' / >
Password:
530 Login incorrect. Login failed.
```

The testing resulted in positive artifacts regarding the successful remote execution of the code. The output of the attacks is presented in **figures 8** and **9**.

It is possible to perform remote command execution when deleting logs and adding users through stealth CSRF attacks injected through the FTP login console. The exploitation can be extended using the advanced codes presented in **figure 10**.

The major risks are of malware infection and remote command execution. The severity of CIAs is high because they infect the server machines on a large scale.

**Figure 8—Leveraging Information Through the FTP Module: Cookie Stealing**

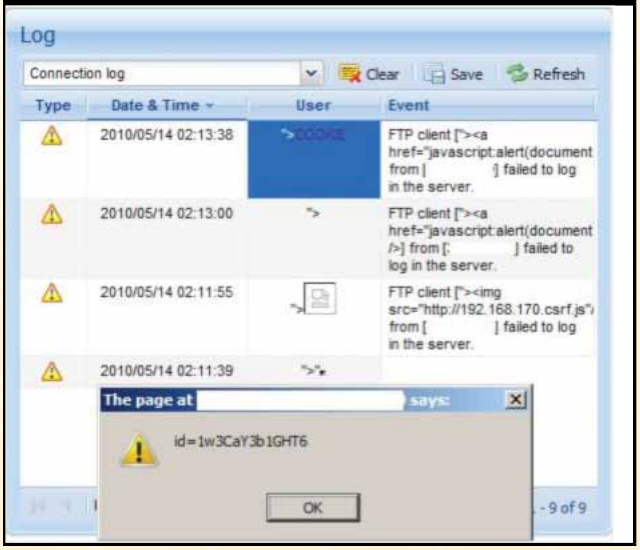
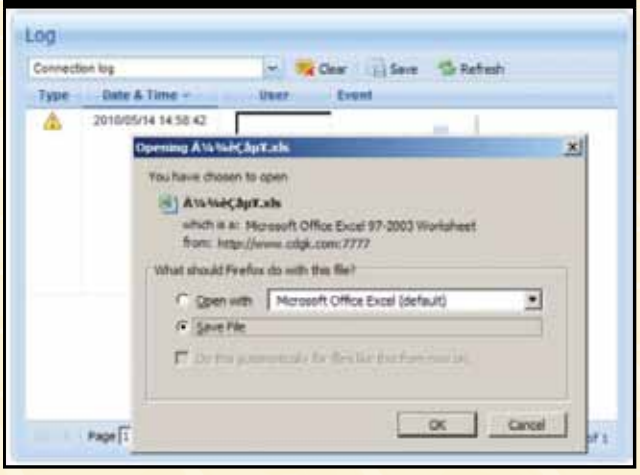



**Figure 9—Successful Injection Leads to Malware Download**




**DEFENSE**

The FTP login consoles or the user verification module should scrutinize the string parameter before verifying the user. A white-list approach should be followed at the protocol level to reduce the impact of exploitation. The applied-design principle should be simple to avoid obscuring vulnerabilities. For example, FTP logs should be rendered in a more customized environment, considering access to a number of clients. The content should be filtered to avoid the use of malicious input, thereby appropriately defining the content type.

**Figure 10—Advanced Code Injections**

*VBScript Code Execution*

```
<object classid='clsid:72C24DD5-D70A-438B-8A42-
98424B88AFB8' id='target' >
</object> <script language='vbscript'>
arg1="c:/WINDOWS/system32/calc.exe"
target.Exec arg1
</script>
```

*Heap Spray Code Execution*

```
var shellcode = unescape("");
var heap_block=unescape("%u0a0a%u0a0a");
var nop_sled= unescape("%u09090%u09090%u09090")
do {
heap_block += heap_block;
} while (heap_block.length < xxxx)
var memory = new Array();
for (ret=0; ret < 100; ret++)
{ memory[ret] += heap_block+nop_sled+shellcode; }
```

## CONCLUSION

This article discussed a unique set of attacks in network devices. The exploitation of a vulnerability depends on the environment, and design plays a critical role in determining the successful exploitation of a vulnerability. Insecurities may prevail in the web interface design of network devices due to lack of security controls. As a result, these devices provide opportunities for attacks. The attacks are more devastating because they result in full compromise of the servers that are providing network services. As discussed in this article, the attack surface depends on several factors, and even a single loophole results in exploitation. The lesson here is that network devices should be developed with due diligence and that secure design principles should be followed.

## ENDNOTES


[1] Sood, Aditya K.; "FTP Anonymous Services—User and Reconnaissance," Security Space for the Untamed Minds, 6 August 2009, *http://zeroknock.blogspot.com/2009/08/vendor-firms-and-anonymous-services.html*

[2] Brown, Jeremy; "Cisco Router HTTP Administration CSRF Remote Command Execution Universal Exploit (2)," SecurityReason.com, *http://securityreason.com/exploitalert/4707*

[3] Lindberg, Henri; Smilehouse Oy; "A-Link WL54AP3 and WL54AP2 CSRF+XSS Vulnerability," SecurityFocus, 31 October 2008, *www.securityfocus.com/archive/1/archive/1/497997/100/0/threaded*

[4] "Siemens ADSL SL2-141 XSRF Exploit," Packet Storm, 26 January 2009, *http://packetstormsecurity.org/files/74295/Siemens-ADSL-SL2-141-XSRF-Exploit.html*

[5] Alme, Christoph; *Web Browsers: An Emerging Platform Under Attack*, McAfee, USA, 2009, *www.mcafee.com/in/resources/white-papers/wp-web-browser-emerge-under-attack.pdf*

[6] Burns, Jesse; "Cross Site Request Forgery: An Introduction to a Common Web Application Weakness, Version 1.2," Information Security Partners LLC, USA, 2007, *www.isecpartners.com/files/csrf_paper.pdf*

[7] Jovanovic, Nenad; Engin Kirda; Christopher Kruegel; "Preventing Cross Site Request Forgery Attacks," *www.seclab.tuwien.ac.at/papers/noforge.pdf*

[8] Barth, Adam; Collin Jackson; John C. Mitchell; "Robust Defenses for Cross-site Request Forgery," CCS'08, 27–31 October 2008, *http://seclab.stanford.edu/websec/csrf/csrf.pdf*

[9] US Computer Emergency Readiness Team (US-CERT), "Vulnerability Summary for CVE-2008-6096," National Vulnerability Database (NVD), 8 March 2011, *http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-6096*

[10] Common Vulnerabilities and Exposures, "CVE-2010-2453," *www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2010-2453*

[11] Branco, Rodrigo Rubira; Aditya K. Sood; "Web Commands Injection Through FTP Login in Synology Disk Station," *www.kernelhacking.com/rodrigo/advisories/CPVDT-2010-9812.txt*

[12] Check Point Software Technologies Ltd., "Update Protection Against Synology Disk Station FTP Login Web Commands Injection Vulnerability," *www.checkpoint.com/defense/advisories/public/2010/cpai-15-Aug.html*

[13] *Op cit*, Sood

[14] Cova, Marco; Christopher Kruegel; Giovanni Vigna; "Detection and Analysis of Drive-by-download Attacks and Malicious JavaScript Code," WWW 2010, 26–30 April 2010, *www.cs.ucsb.edu/~vigna/publications/2010_cova_kruegel_vigna_Wepawet.pdf*

[15] Egele, Manuel; Engin Kirda; Christopher Kruegel; "Mitigating Drive-by Download Attacks: Challenges and Open Problems," *http://iseclab.org/papers/inetsec09.pdf*

[16] *Op cit*, US-CERT