Writing interesting CTF Services and good testscripts

Hans-Christian Esperer, CDA
hc@hcesperer.org

2. Oktober 2008
1. Writing interesting CTF Services
   - Common misconceptions
   - Design principles
   - Some examples

2. Writing good testscripts
   - Overview
   - Formalities
   - Using the testscript template

3. Using the gameserver
   - Installation
   - Configuration
   - The game
Common misconceptions

- Writing a CTF service is a neat programming exercise
- Writing bad code will „automatically add“ vulnerabilities
- Choosing an unknown programming language does the trick
- Using an exotic unix flavor makes a CTF more interesting
- Leave the testing to the teams
- „The hard work is done, now there’s just the testscript left“
Intimate language knowledge leads to better services

„Good“ vulnerabilities can only be added to nicely written code

(Ab)using features of exotic programming languages makes a CTF more interesting

Use a platform your services run well on

Well tested services are good for the organizers, for the teams and for your karma ;-)

Good testscripts are essential to the CTF’s success
How to begin

Some easy steps to follow...

- Idea for a service
- Suited as a CTF service?
  → Can it be used to store and retrieve (key, value) pairs? (Flags)
  → Yes? Good.
- Implement the service
- Write the testscript, test it with the service
- Add vulnerabilities
- Test it again
- . . . and again!
Design principles for CTF services

- Implement useful functionality
  → Don’t think of flags, think of real world data
- Keep code and protocol quality high, but
- Violate the RFCs twice
- Write stable code
- Write secure code first, add vulnerabilities later
- Good code is well tested code
- Vulnerabilities must be *in* your code, not in 3rd-party libs
- Be aware of your vulns’ impact
- Mix easy and hard to find vulnerabilities
Writing a CTF service is not a programming exercise.

- Write your service in a language you know well
- However, you will learn a lot about your system
The simplest service ever

Two important things every service must be able to do:

- Accept a *key* and a *value*
- Send the *value* when *key* is requested

For most services, both actions are very complicated
The simplest service ever

```
hc.hc-laptop ~/da-op3n010/trunk/testscripts/connectivity> ./message.py store localhost foo bar
Sending bar to localhost
Service responded according to protocol.OK
hc.hc-laptop ~/da-op3n010/trunk/testscripts/connectivity> ./message.py retrieve localhost foo bar
using port 1985
bar
Flag successfully retrieved!
hc.hc-laptop ~/da-op3n010/trunk/testscripts/connectivity> 
```

```
hc.hc-laptop ~/da-op3n010/trunk/testscripts/connectivity> python mb.py
Waiting for connections...
foo -> bar
sending foo to 127.0.0.1:1985
```
How flags were stored and retrieved...

- IRC daemon with ChanServ and NickServ written in python
- Flags were stored by registering channels
  → FlagID: Channel name, Flag: Channel topic
- STORE: testscript joins a channel, registers nick and registers channel
- RETRIEVE: testscript joins channel, waits for chanserv to set correct topic
How you could steal flags... 

- Join #irclogs, get a message for every registered channel
- Become operator (default password), then use ChanServ listall command
- Use broken MODE command to become operator
- +s chanflag was not respected → repeatedly call LIST, until a flag-channel is discovered
- Use SQL injections (simple injections didn’t work, though)
- Bypass various broken permission checks
- Become super operator using broken OPER command, then inject python code
  → Fix the OPER command, do not remove the injection functionality
Writing good testscripts

- Writing good services is important, but
- Writing good testscripts is crucial
- It’s pretty hard
- Actually, it isn’t, but it takes time ;-) 
- Think like a mathematician working on a proof
  → Nothing is guaranteed, unless you ensure it is
Requirements

Testscripts must be much more flexible than services.

- Write platform independent code
  → You don’t know where they will run
  → Distributed execution possible
- Test your code on various platforms
- Produce debug output
  → very important for gamemasters
- Deal gracefully with all errors
- Write efficient code
  → $n \cdot m$ processes in parallel (max), $n ==$ num. of teams, $m ==$ num. of testscripts
- Write secure code
Write secure code

- Gameservers are off limits
- Teams still try to exploit them
- Erraneously fixed services may send garbage
- Erraneously fixed services are unpredictable
- Secure code is stable code
- Testscripts must be stable
- CTF is about security, so set a good example ;-)
Formalities

Important rules for testscripts

- Do not fork
  → Threading is OK
- Set memory limits
- Each testscript can run $n$ times in parallel, $n = \text{number of Teams}$
- Guarantee: Only one testscript per (team, service) at any time
- Maximum runtime: 60 seconds, after that: KILL -9
- Minimize startup overhead
  → Do not use Java, C#,... to code your testscript
  (Coding a service in Java is perfectly OK)
- Testing must be stateless
- Gameserver / Testscripts have no special privileges
Stateless testing

- Teams may “reset” their vulnimages at any time
  → Testscripts must still work!
- It is OK to create accounts, but
- Do not assume an account lives forever
  → If an account doesn’t exist anymore, create a new one
- **Test** that this actually works!
No special privileges

The Gameserver / the testscripts have no special privileges

- Test normal service functionality
- There are no STORE FLAG / RETRIEVE FLAG functions!
- Flags are normal data
  - Bank account data
  - IRC Channels/topics
  - S.*VZ userdata
- If an account is needed, the testscript must create it
- No usage of admin accounts
If your STORE routine fails, the flag is discarded
Discarded flags cannot be captured
But: teams want to capture flags
→ If there is a slight chance a flag reached the service, do not discard it
Probe for additional functionality during RETRIEVE, not STORE
There’s an extra error type: Service lacks functionality
HC’s testscript template ;-)

- Testscript template in python
- Works with hc’s gameserver ;-)
- Common structure for all testscripts is good
- Python is nice and quite platform independent
Testscript template ;-) 

- Use `die(REASON)` to quit. Always
- Use `LineReader(SOCKET)` to create a limited line reader
  → LineReader reads up to 8192 bytes, then exits with an overflow error
  → Adjustment needed if your service sends more data, (8192 reasonable default)
- Use `except(l, regexp, error)` to wait for an answer
  → l: LineReader, regexp: what to look for, error: what to throw if regexp isn’t read in time
- Use `randflag()` to generate an invalid flag
  → Invalid flags are indistinguishable from valid flags
- Write your testing code in `store()` and `retrieve()`
Testscript template: STORE routine

```python
#!/usr/bin/env python2.5

def store((ip, flagid, flag)):
    seed(time())
    s = getsock()
    try: s.connect((ip, 25))
    except: die('conn')  # Connection refused or timeout
    s.settimeout(10.0)
    lr = LineReader(s)
    if random() < 0.5: pass
    else: pass
    (hostid, bar) = expect(lr, '^[0-9]{3} (.+)ESMTP(.+)', 'greeting')
    s.sendall("HELO hcesperer.org\n")
    sys.stderr.write("Host identified itself as %s\n" % hostid)
    sys.stderr.write("====== SUCCESSFULLY STORED FLAG ======
")
    die('allok')

# vim:set shiftwidth=4 tabstop=4 expandtab textwidth=79:
```

Hans-Christian Esperer, CDA
Testscript template: Error codes

```python
#!/usr/bin/env python2.5
ERR_OK = 0   # All OK
ERR_CONNECTION = 1   # Connection refused / connection attempt timed out
ERR_WRONGFLAG = 5   # A wrong flag / no flag was returned
ERR_FUNCLACK = 9    # The service lacks functionality
ERR_TIMEOUT = 13    # Done automatically by the scorebot under normal circumstances
ERR_UNKNOWN = 17    # Temporary status – do not use unless you’ve got a very good reason
ERR_GENERIC = 21    # Be sure to include a descriptive message if using this error
ERR_PROTOCOL = 25   # Protocol violation

shit = {'conn': ('Unable to connect to the service', ERR_CONNECTION),
        'greeting': ('The server didn’t greet correctly', ERR_PROTOCOL),
        'allok': ('Everything is fine', ERR_OK)}
```

```bash
#vim:set shiftwidth=4 tabstop=4 expandtab textwidth=79:
```
Writing interesting CTF Services
Writing good testscripts
Using the gameserver

Installation
Configuration
The game

Prequisites

- PostgreSQL 8.1 or later
- python 2.5 or later
- bzip2
- JRE 1.6 or later
- telnet
- → telnet readline extension (ask alech ;-)
CTF Gameserver configuration

Visit http://ctf.hcesperer.org/gameserver/installation.html
Rating advisories

- `accept(advisoryID, pointsToAward, comment)`
  → `accept(1, 2, "Good one, two points!")`
- `reject(advisoryID, comment)`
  → `reject(2, "Write beter English next time!")`
- `delete(junkID)`
  → Use delete **only** to delete **junk**.
Upcoming CTFs

25c3-CTF  http://ctf.hcesperer.org/25c3ctf
USCB  http://google.com/search?q=uscb+ctf ;-)
CIPHER 5  http://www.cipher-ctf.org/
→ MRMCDs111b 6. September 2008 in Darmstadt