SSRF pwns: new techniques and stories

@ONsec_lab: http://lab.onsec.ru

Alexander Golovko
Vladimir Vorontsov
SSRF pwns: new techniques and stories

@ONsec_lab: http://lab.onsec.ru

Alexander Golovko
Vladimir Vorontsov
About us

ONsec - web application security company founders since 2009

Alexander - network security expert, Debian GNU/Linux maintainer

Vladimir - webapp security expert, bughunter

@ONsec_lab - webapp security research

Twi+Blog: http://lab.onsec.ru [ENG]

Awarded by Google, Yandex, bla-bla-bla

Wants to create yet another Web App Firewall ;)

ONSEC.
About SSRF

First described in 2008, Deral Heiland http://goo.gl/Q5ZDh
Reincarnated for XXE during Yandex's Month of SecBugs
(end of 2011), @ONsec_lab: http://goo.gl/9OXfu

Exploited SAP through gopher in 2012, BH-US: http://goo.gl/Lt4pr, ERPScan - A.Polyakov, D.Chastukhin


Exploited memcached, fastcgi, etc: http://goo.gl/D8UCd

Top Ten Web Hacking Techniques of 2012 2nd place:
http://goo.gl/XUWS8 "Pwning via SSRF (memcached, php-fastcgi, etc)"

CWE-918: http://cwe.mitre.org/data/definitions/918.html
About SSRF

What is Server-Side Request Forgery?

CWE-918 not so correct:
The **web server** receives a **URL** or similar request from an upstream component and retrieves the contents of this URL...

Not only web-servers, not only URL

```php
fputs($f,"GET /index.php?username={$_POST['login']}
HTTP/1.1\r\nHost: $host\r\n\r\n"); //CRLF injection
```
Before we start

SSRF for bypass host-based auth
SSRF for bypass firewalls
SSRF for bla-bla-bla-bla

But is there any other ways to do the same?
Hello from early 90th!

Packets forwards between interfaces
By default in Debian/RedHat
UDP packet can be easily sent from Internet, classic spoofing (DDoS way)

Can exploit your SNMP, memcached, others UDP+host-based auth servers
Use sysctl net.ipv4.conf.<all>.rp_filter
Advanced UDP spoofing exploitation

Exploit services as SSRF where response is request to another service
Ping-pong SSRF, spoofing based SSRF

Firewalls bypass in deep network by chaining requests, no restrictions more!
Reflection SSRF attack

- Spoofing attack where service response used as a request for another service - Server-Side Request Forgery
- In spoofed packet attacker set source IP/port from victim
- Memcached easy to be exploited
- Echo service is ideal for this purpose
Reflection SSRF attack

Spoofed packet, source address: Host B

Host A
Service A

Firewall

Impossible

Response for spoofed packet

Host B
Service B
"Ping-pong" effect (UDP)

1. Spoofed packet:
   set key 1 3600 5 stats

2. Spoofed packet, source address:
   Host B get key

3. Host A memcached

4. 0x01 ... VALUE key stats
   0x01 ... ERROR STAT PID ..
   ...

5. ... infinite loops ...

By default memcached listened at :11211 TCP and UDP both!
"Ping-pong" effect (UDP) exploit

```
sudo packit -t udp -s 10.3.0.5 -d 10.3.0.4 -S 11211 -D 11211 -p '0x 01 01 00 00 00 01 00 00 67 65 74 20 61 61 0d 0a'
```

Request for "aaa" key value
Value of "aaa" is "version"

Execute commands:
"VALUE aaa 0 14", than "version"
Results: "ERROR" and "VERSION 1.4"

Ping-pong infinite loops
ERROR
ERROR
...
Hello from 2012!

TCP Fast Open (since kernel 3.6)

Provide SYN+data packets

Required Cookie

Cookie = AES(key,ClientIP)

Key have 16 bytes length

One key for all clients

UNBRUTABLE :(((

waits for others TFO impl-s
TCP Fast Open

By design security limitations:
One cookie for a one client, ports are not restricted
One secret key for a server, for all clients AES(key,IP)
Hello from 2012!

IPv6 configuration issues for SSRF!

1. Bypass simple filters by ::1
   http://::1/server-status ;)
2. Link-local firewalls bypass
3. Exploiting autoconf IPv6
TCP Fast Open attack concept in clouds

1. Timestamp A
   TFO SYN with cookie request
   TFO cookie for IP 10.3.13.37

2. Timestamp B>A
   TFO SYN+data+old cookie for IP 10.3.13.37 spoofing Host C
IPv6 link-local addresses

Hosts A and B are in one network segment

Host A (already hacked)

1. Firewall block

2. No firewall rules for link-local IPv6 address

- Link-local address can be
  - sniffered (root required)
  - calculated by MAC:

Host B
Service B Listen *:80
IPv6 Router Advertisement

Hosts A and B are in one network segment

1. Firewall blocked :80
2. RA packet with new IPv6 address
3. No firewall rules for new IPv6 address

Host A (already rooted)

IPv6 autoconf is enabled by default in Debian/RHel

To disable use sysctl net.ipv6.conf.*.autoconf

Host B
Service B Listen *:80
What's the conclusion?

Host-based auth must die!
Now we start


SSRF not only in webapp code now, i.e. "ping-pong" attack and UDP memcached example of it
Protocol schemas

Different protocols = different actions
Not only sending data, but data leak also

See "SSRF bible. Cheatsheet":
Exploitation->Original request data sniffing (http://goo.gl/oRMhg)
Protocol schemas

telnet:// protocol schema

● read data from stdin
● write data to stdout

what are stdin/stdout for your webapp?

For CGI - HTTP request/response
For mod_php, FCGI - /dev/null ;(
CGI is still for Enterprise webapps ;)

ONSEC.
Which server is the most secure in your environment?

VPN? Other SSL server?
SSL -> PKI -> SSRF !!!

Client certificate ----->
OCSP/TSP/CRL URIs ----->
OCSP/TSP/CRL requests
SSRF on PKI

Public Key Infrastructure
Client certificate validation
External resources defined in certificate, such as CRL, OCSP, TSP urls
Certificate validation logic is different by implementations
Different implementations

- Check CRL/OCSP url from config, not from user certificate (nginx)
- Check trust relationship before certificate status
- Check certificate status before trust relationship (CA, intermediate)
- Check intermediate/CA certificate status before trust relationship
Different implementations

- Parse certificate
- Is cert self-signed?
- Verify certificate status
- Verify ...

- Parse certificate
- CA (intermediate) validation process
- Verify CA (intermediate CA) status
- Verify ...

- Parse certificate
- Verify trust relationship (CA/intermediate/client certificate)
- Verify ...

SSRF!!!
SSL -> PKI -> SSRF

CRL: dict://hostB:11211/1stats
OCSP: dict://hostB:11211/1stats
TSP: dict://hostB:11211/1stats

Host A
SSL (https, VPN, etc)

Firewall

Host B
memcache *:11211
SSRF practice. Yandex

Something interesting?
- Exploited memcached through SSRF
- Discovered few intranet services
- Discovered infrastructure bugs
- Got fun and skills
- Shocked yandex security team :)}
SSRF practice. Yandex

11 SSRF bugs accepted
7 XXE + SSRF bugs accepted
~ $12900 total reward
~ $760 per bug ($1000 max award by program)
Nice SSRF using DNS ;)

- Webmaster service provides content receiving of YOUR sites
- Validation process based on files/DNS
- Verification by DOMAIN, not by IP
- Attack vector: verify domain, then change A-record to Yandex's intranet
- Profit!
Nice SSRF using DNS ;)

Intranet content ;)
Yandex SSRF discovery

- Intranet scan using SSRF is not ethical
- Using Google to find Yandex's intranet hosts is so ethical ;)
- Exploitation of SSRF to retrieve sentences data is not ethical
- Impact must be demonstrated to bug reviewers
Using Google to hack Yandex ;) 

Googled config with IP and domain: