Advanced Exploitation Techniques

Overview

How Do Exploits Work?

Lets look at a few types of exploits!

- Format Strings
- Race Conditions
- Heap Spraying
- Heap Overflows
- Buffer Overflows

The Metasploit Project

Saint Exploit – Cost Effective Choice

Core Impact In-depth

How Do Exploits Work?

An exploit refers to a malicious computer attack that takes advantage of a vulnerability, bug, glitch, or security hole that can lead to privilege escalation or denial of service on a computer system.

Exploits can also be classified by the type of vulnerability they attack; including buffer overflow, format string attacks, race condition, and cross-site scripting.

Many exploits are designed to provide root level access to a computer system. However, it is also possible to use several exploits, first to gain low-level access, then to escalate privileges until one reaches root.

Blackhat hackers do not publish their exploits but keep them private to themselves or other malicious hackers. Such exploits are referred to as 'zero day exploits' and to obtain access to such exploits is the primary desire of unskilled malicious attackers, so called script kiddies

Format String

They can be used to execute an attacker's code, either causing a DoS or taking control of the system.

The issues: An attacker can use unfiltered user input as the format string parameter in certain C functions that perform formatting, such as printf().

- You could print data from the stack or command printf() and similar functions to write the number of bytes formatted to an address stored in the stack.

Most exploits will use a combination of techniques to force the program to overwrite the address of a library function or the return address of the stack with a pointer to some malicious shellcode.

Race Conditions

A Race Condition is when a hacker can take advantage of a program while it is performing a privileged operation. This occurs in a multi-processing system.

- **It is a classic case of the left hand does not know what the right hand is doing.**
- **The output and/or result of the process is unexpectedly and dependent on the sequence or timing of other events.**
- **Race conditions arise in software when separate processes or threads of execution depend on some shared state. Operations upon shared states are critical sections that must be atomic to avoid harmful collision between processes or threads that share those states.**
- **Example- wu-ftd 2.4 signal handling vulnerability**

Memory Organization

The basic exploitation techniques can be methodically categorized. You must be aware of the basic process of memory organization. A process running in memory has the following sub-structures:

Code is the read-only segment that contains the compiled executable code of the program.

Data is writable segments containing the static, global, initialized and un-initialized data segments and variables.

Stack is a data structure based on Last-In-First-Out ordering. Items are pushed and pulled from the top of the stack. A Stack Pointer (SP) is a register which points to the top of the stack (in most cases).

Heap is basically the rest of the memory space assigned to the process.

Buffer OverFlows

One of the biggest security risks ever.

This technique of exploitation is straightforward and lethal.

A program's stack stores the data in order, whereby the parameters passed to the function are stored first, then the return address, then the previous stack pointer and subsequently local variables.

If variables (like arrays) are passed without boundary checks, they can be overflowed by sending into them large amounts of data, which corrupts the stack.

This leads to the overwrite of the return address and consequently a segmentation fault. If the trick is craftily done, you can modify the buffers to point to any location, leading to code execution.

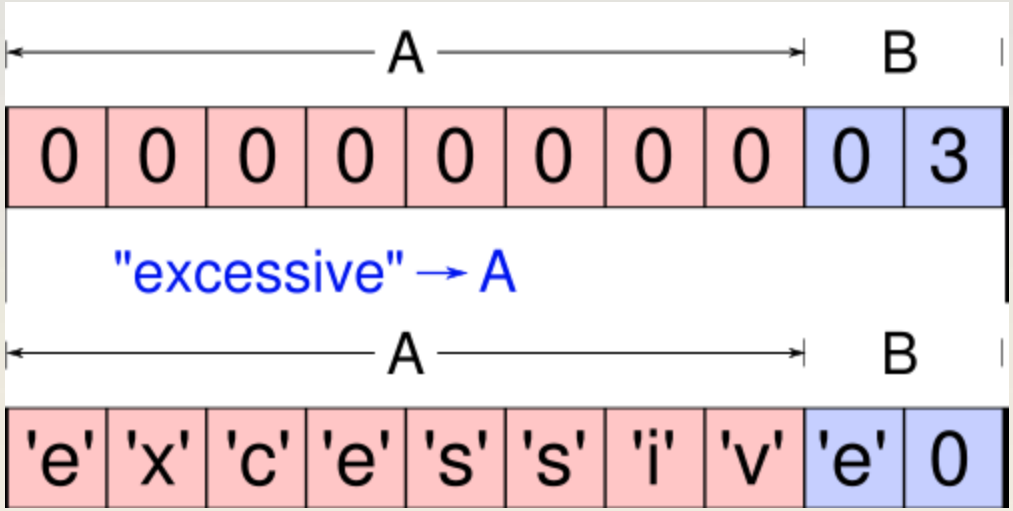
Buffer Overflow Definition

A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code.

From http://www.owasp.org/index.php/Buffer_Overflow

In 1996, the European Space Agency's *Ariane 5* rocket exploded right after launch because a program tried to put a 64-bit number into a 16-bit memory space.

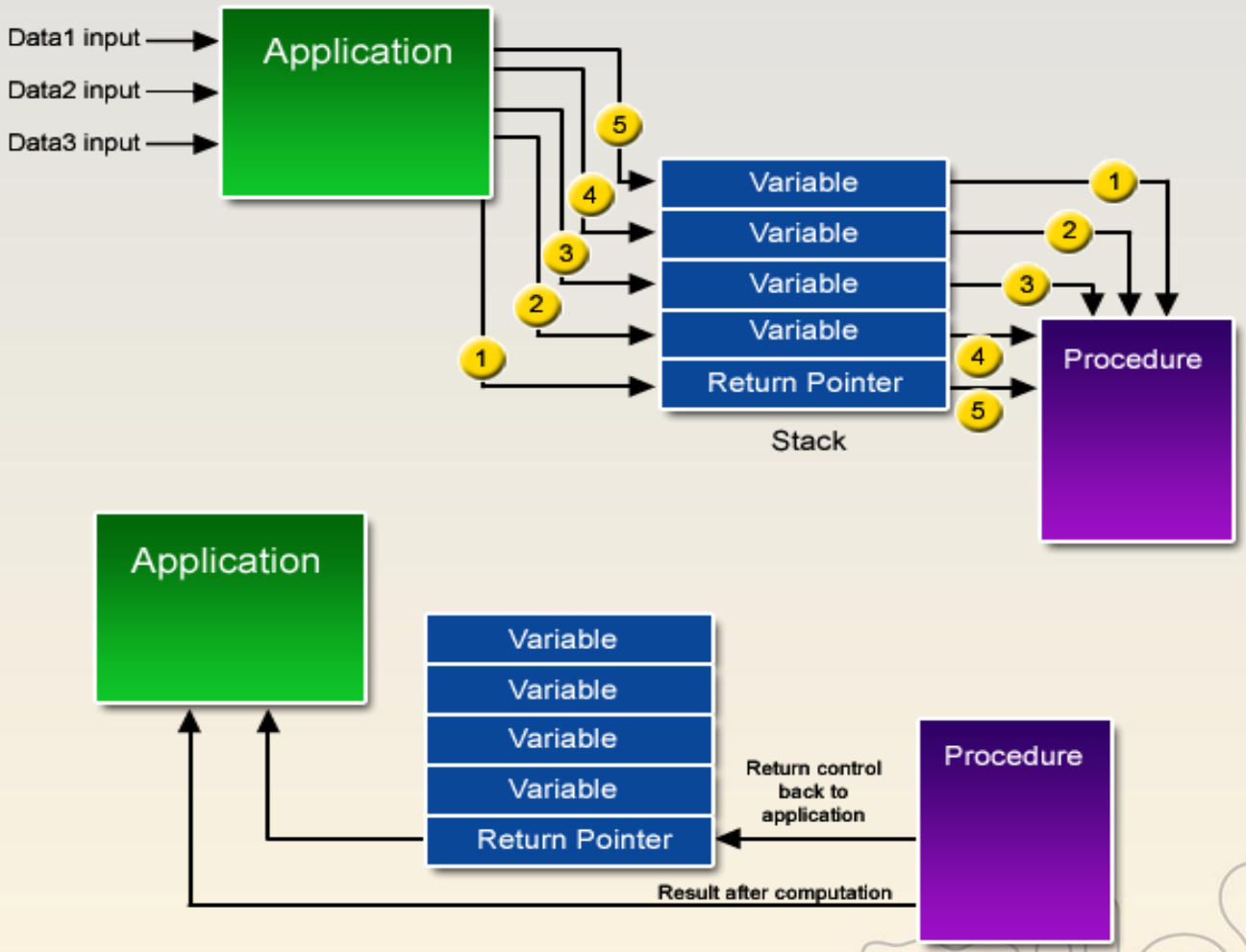
Overflow Illustration



This is a basic example of a buffer overflow. By writing 10 bytes of data into "A", which only has 8 bytes available, "B" is changed unintentionally.

Image and text from:
http://commons.wikimedia.org/wiki/File:Buffer_overflow_basicexample.svg

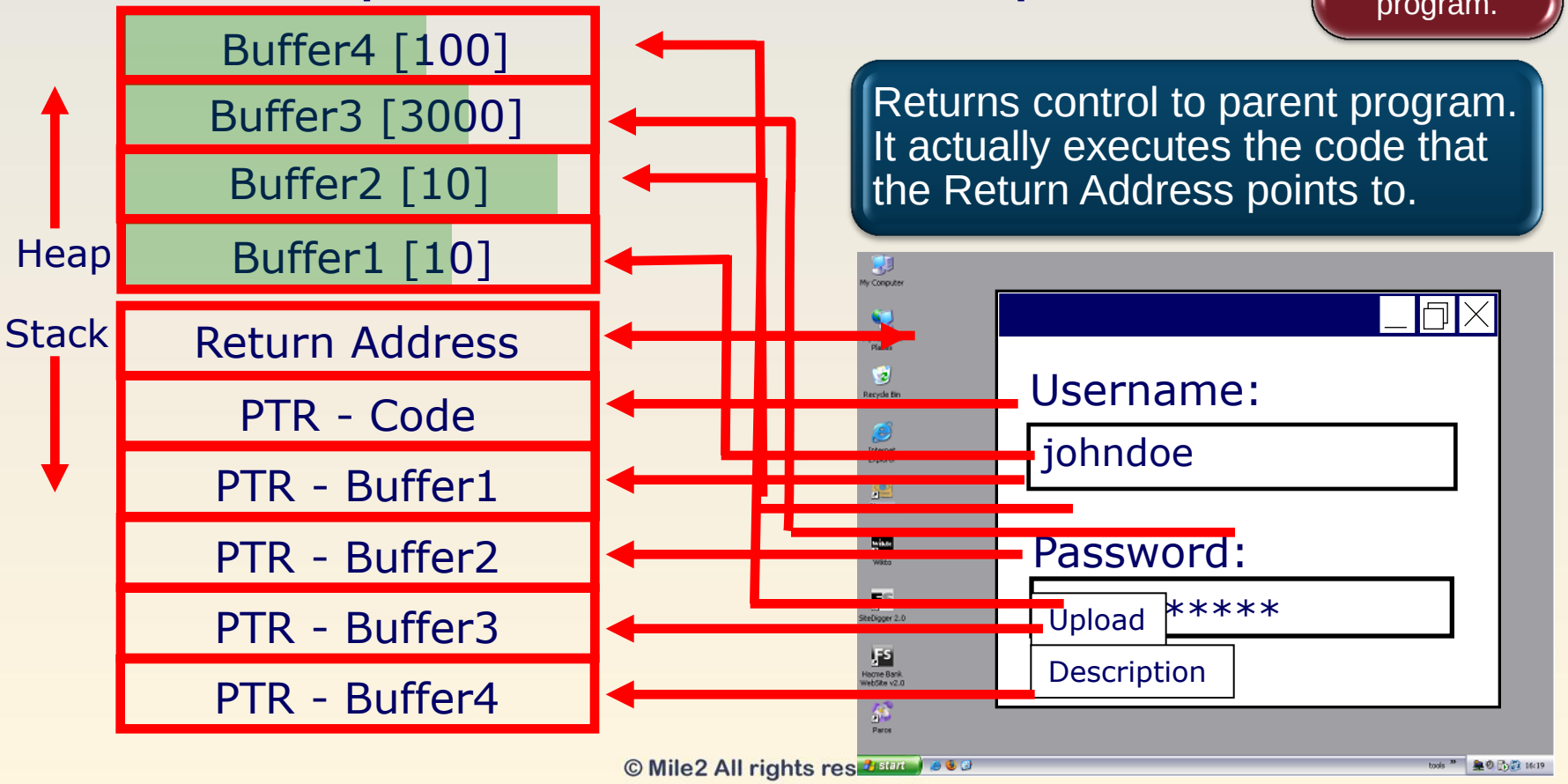
How Buffers and Stacks Are Supposed to Work



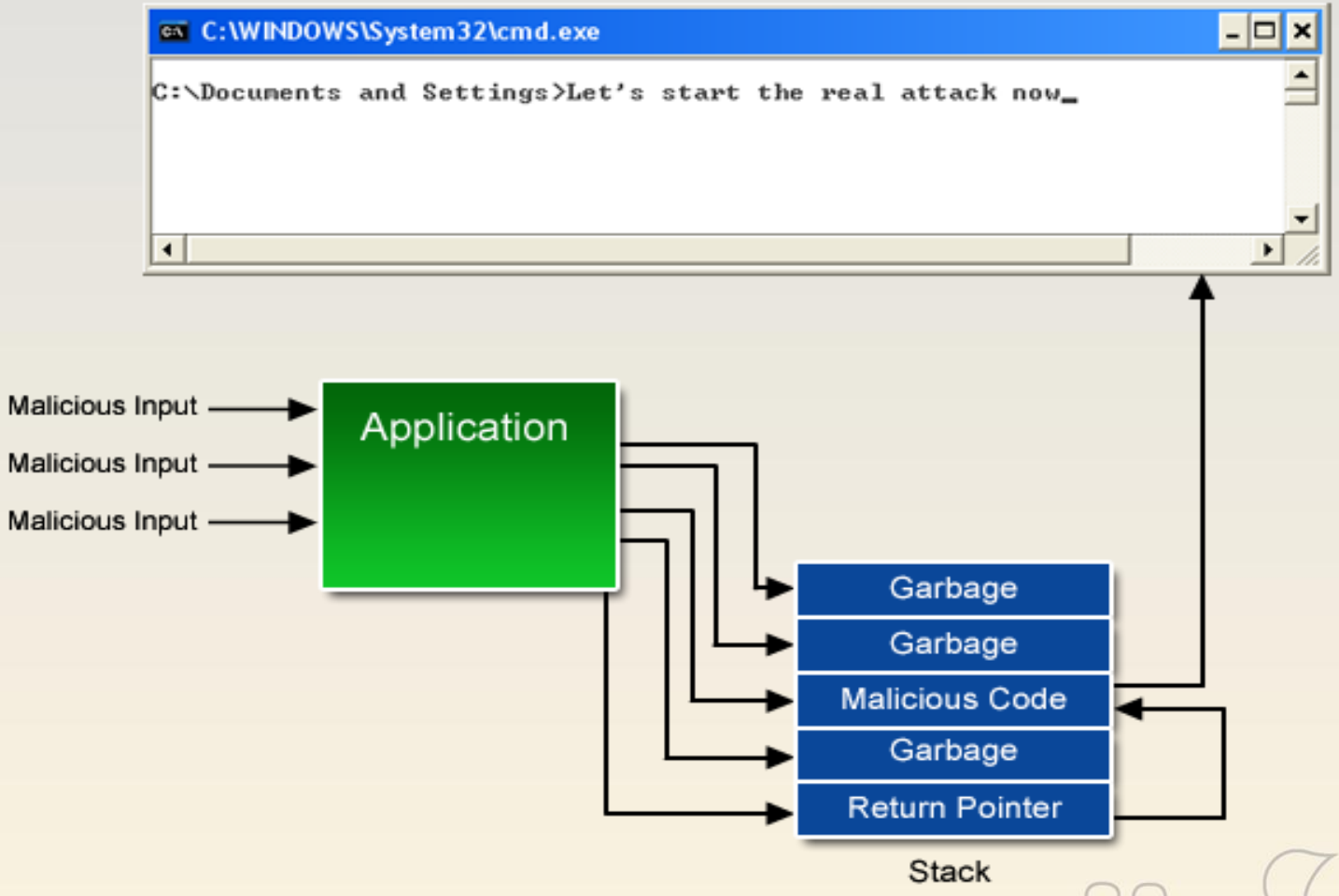
Stack Function

- **The below example shows the stack and heap entries for a simple program.**
  - **A user will enter a username and password.**
  - **Then upload a file and write a description.**

This is an example of how the stack works while using a program.



How a Buffer Overflow Works



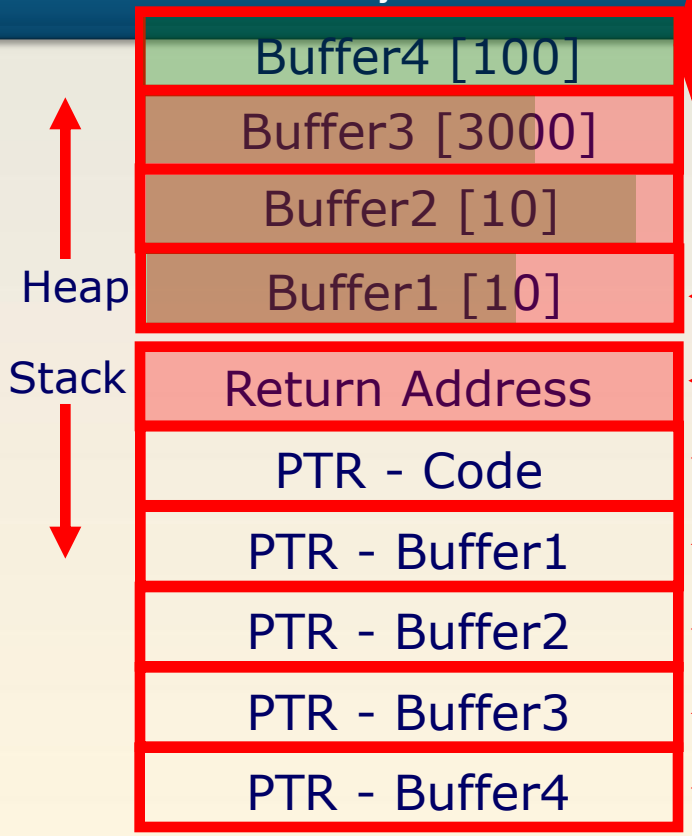
Buffer Overflows

To overflow the buffer, you must calculate an entry point to the program and work out exactly how much data to input so that it overwrites the return address.

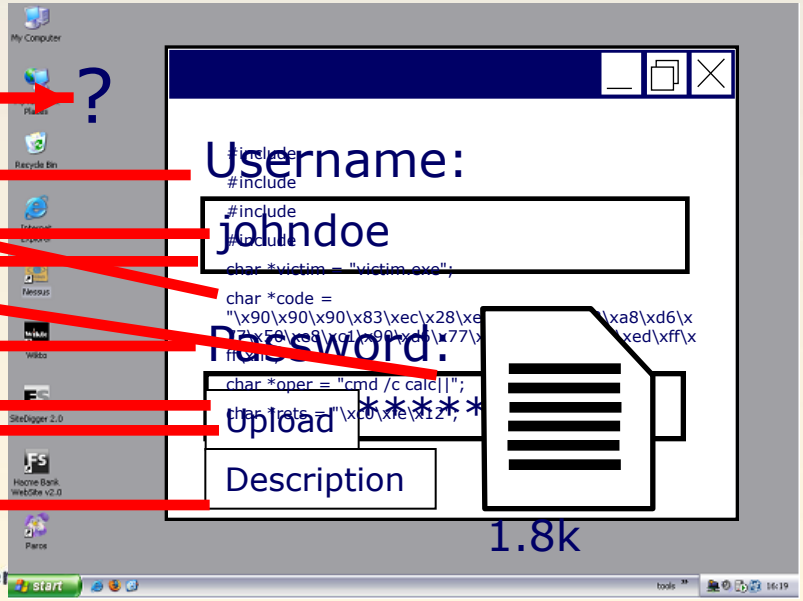
- Remember that the return address tells the OS where to go next to execute code.

If it has been overwritten with malicious data, it could point somewhere else i.e. the beginning of the inputted data.

This then executes the injected code and exploits the machine.



Returns control to the code that the Return Address points to, the address that the hacker has changed!



Heap Overflows

It is when information that is used to link memory together is corrupted.

- **Example: When a function uses the first 4 bytes of each chunk to link the next chunk of memory. If information is overflowed from one chunk to an adjacent address, it would overwrite this linked list mechanism. It would allow the linked list to jump to whatever memory address is linked to the overwritten value.**

The attacker can then run his code!

Heap Spraying

This is used for arbitrary code execution.

It is basically code that sprays the heap and attempts to put a certain sequence of bytes at a predetermined location in the memory of the program or process being attacked.

The code will have the memory allocate blocks in the process' heap and fill them with values of its choosing.

Prevention

Change the Culture – Integrate Secure Software Development into the process. (SDL)

- Encode the secure development into your policies.
- Measure your effectiveness.
- Establish an accountability model for security.
- Appoint a security liaison.

Educate both the developer and the end user.

Utilize Threat Modeling.

Create or use Code Checklists.

Perform Security Testing.

Patch The Operating System And Application as patches are leased.

With regard to technology.

- Migrate your software products to managed development platforms such as Sun's Java or Microsoft's .NET Framework.
- Utilize an Input Validation Library.
  - http://www.microsoft.com/technet/security/tools/urlscan.mspx
- Watch for new technology developments.

Security Code Reviews

What is a security code review?

Process in which code is reviewed for flaws that could compromise the confidentiality, integrity or availability of a system

The objective is to:

- Catch as many problems as possible
- Educate others on how to write secure code and how to conduct secure code reviews
- Strengthen your knowledge of the application

Secure Code Review

The Secure Code Review Process

- Know the Vulnerabilities
- Know the Business Risks
- When to conduct the code review
- Who should be involved
- What to look for
- Fixing the Issues
- Using Automated Tools

Secure Code Review

Know the Vulnerabilities

- In order to find a security vulnerability, you have to know what one is
- **How do you accomplish this?**
  - Attend training
  - Read, Read, Read!!!!
    - There is a wealth of knowledge published on known vulnerabilities and ways to protect your systems
  - **Practice**
    - It is important to learn by doing.
      - Pair experienced reviewers with junior reviewers to spread the knowledge

Secure Code Review

Know the Business Risks

- **Know what assets are being accessed by the code and the level of protection that is required.**
  - Will need to understand what is being protected to know if your code is protecting it.
- Review use cases:
  - Do different types of users take different paths
  - Do a walk through at the code level using the use cases to structure your review

Secure Code Review

When To Conduct the Code Review

- The perception of time constraints is usually why code reviews are not done
- Remember, fixing security issues after deployment can be costly and more time consuming
  - The deployment process will have to be repeated for issues found in production
  - How long can you allow insecure code to be exposed to the public?
    - Business decision?

Secure Code Review

When To Conduct the Code Review

- It is important to have a target when doing the review
  - **What components have the largest attack surfaces?**
    - What components are touched by the most users/process?
      - Web facing services?
  - **What components protect the most important data?**
    - Database access code
    - Encryption components
    - Session management code
    - What is most important to your company?

Secure Code Review

When To Conduct the Code Review

- **Will time allow it?**
  - Will time allow reviews after major components are complete?
  - Will time only allow a review after the code is complete?
    - Remember it's always better to find the bugs as early in the process as possible
    - If you wait until the code is complete and issues are found, more code may have to be changed
- **Put some process in place**
  - Any review that finds issues is better than no review at all
    - Do what you can...

Secure Code Review

Who Should be Involved

- Developers who are familiar with the architecture
- Strong Developers
- Developers with knowledge of security

- Knowing the architecture has the advantage of knowing how components SHOULD communicate
- Who are your strong developers?
- Pair them with junior developers?
- Strong developers with knowledge of secure coding practices adds value of knowing how code should be written securely

Secure Code Review

What To Look For

- Configuration
  - Many configuration files have default settings that have known security flaws
  - Is there any sensitive information in the configuration files?
    - Db usernames/passwords
    - Test IDs?
  - Protect files according to what resources they control

Secure Code Review

What To Look For

- Authentication
  - Is strong authentication being used?
    - Brute force attacks and Dictionary-based attacks
      - Attacks that attempt to guess login credentials by escalating known credentials or using common words
        - Example:
    - Multi-factor authentication
      - Example: Attacker would have to guess the login credentials and have a token to log in
  - Are account lockouts being used and enforced?
    - What is the process for unlocking accounts?

Secure Code Review

What To Look For

- **Logging**
  - Are all login attempts being logged?
    - Remember Intrusion Detection
  - Is logging centralized?
    - Using a centralized approach encourages consistency and code re-use
  - What is being logged?
    - Is the application logging sensitive information?
      - If so, what protection mechanisms are in place to prevent the log files from disclosure
    - Does the log file allow scripting tags to be written?
      - Think cross-site scripting...

Secure Code Review

What to Look For

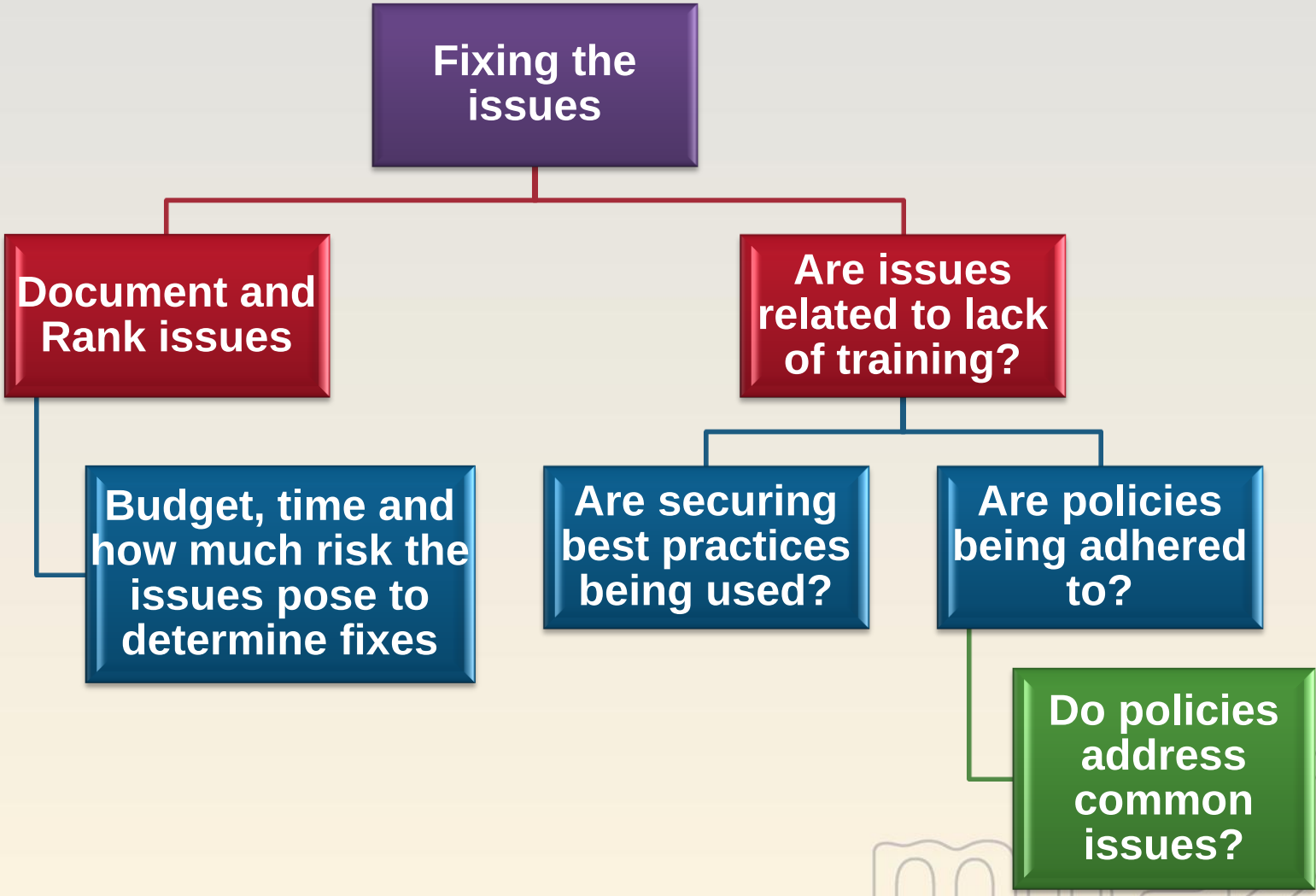
- Error and Exception Handling
  - How are errors and exceptions handled?
    - Does the application return any sensitive information to the user?
      - Detailed error messages...etc
        - Enumeration
  - Ensure that all sensitive operations are wrapped appropriately
    - Database methods
    - Encryption processes

Secure Code Review

What to Look For

- **Data Validation**
  - Weak validation is usually the reason most attacks are successful
    - Is the application validating the data for the following:
      - Type
      - Format
      - Length
      - Range
      - Valid business values
  - Data should be validated before constructing SQL statements
  - Validate that output does not contain scripting characters

Secure Code Review



Secure Code Review

Automated Tools

- Can be used to find common issues
  - They normally don't find complex issues:
    - flaws in encryption algorithms
    - flawed business logic
- Consider using both an automated approach and manual review:
  - Catch low hanging fruit with automated tool
  - Catch more complex issues with manual review

Stages Of Exploit Development

Shellcode

- A shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine.

Injection vector

- The pointer or offset where the shellcode is placed in a process and the return address is modified to point to. The whole point of the Injection vector is to get the payload (shellcode) to execute.

Request builder

- This is the code which triggers the exploit. If it's related to string functions, then scripting languages are generally preferred.

Handler Routine

- This part generally consumes the majority of the code. This is a handler for the shellcode performing operations such as linking to a bindshell.

Stages of Exploit Development

User options handler

- A front-end that provides the user with various control options like remote target selection, offset selection, verbosity, debugging and other options.

Network connection Handler

- This comprises of the various routines which handle network connections like name resolution, socket establishment, error handling etc.

Shellcode Development

It is the code used as the payload in the exploitation.

You can add new payloads in Metasploit.

- An exploit payload is that arbitrary code used after the exploit gains the capability to execute code.
- An auxiliary payload is not necessarily used with an exploit and contains functionalities like port scanning . Examples are DoS, fuzzers and other reconnaissance tools.

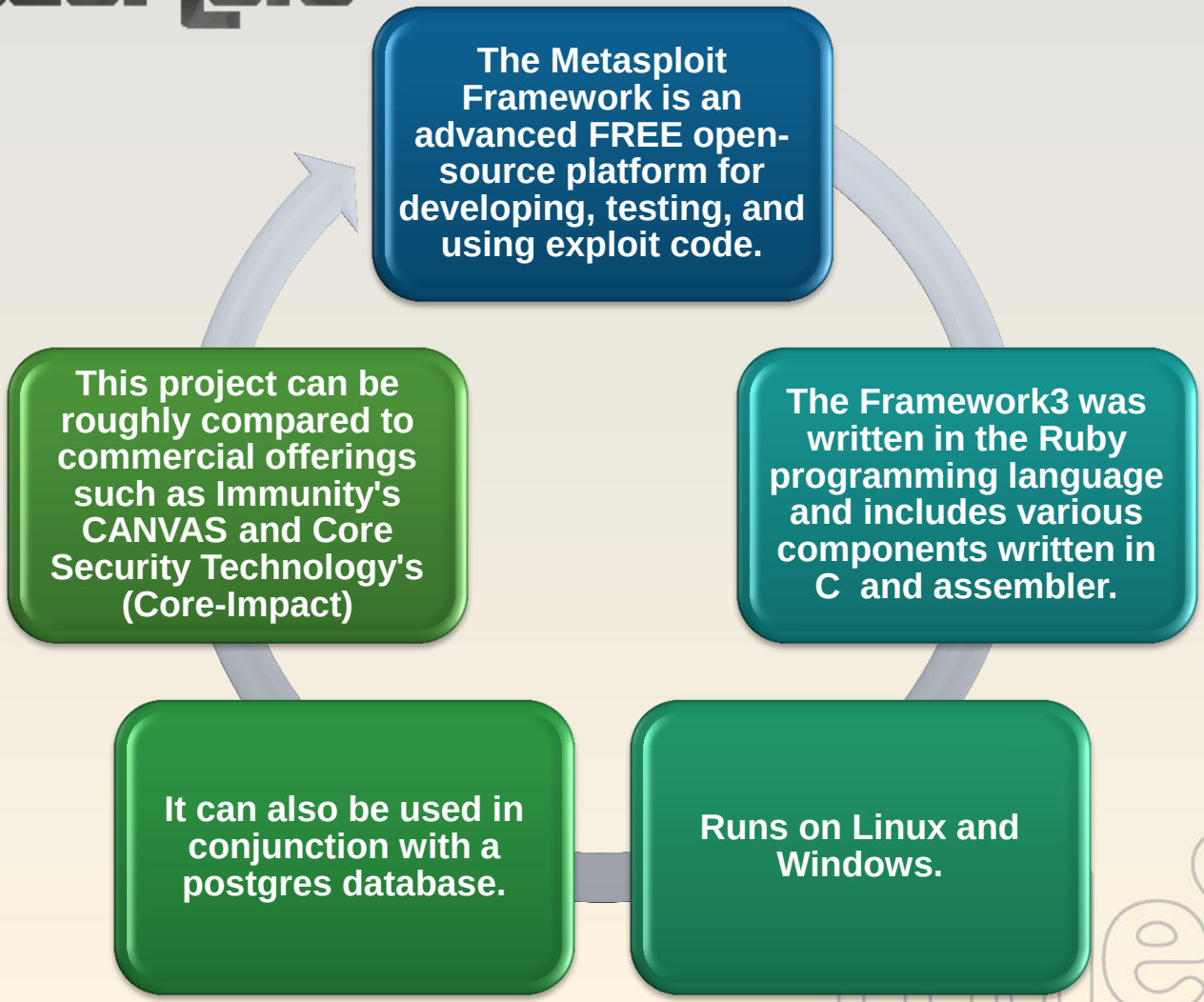
Types of Shellcode:

- Local – used often for privilege escalation.
- Remote – most common, provides access to the target machine from across a network.

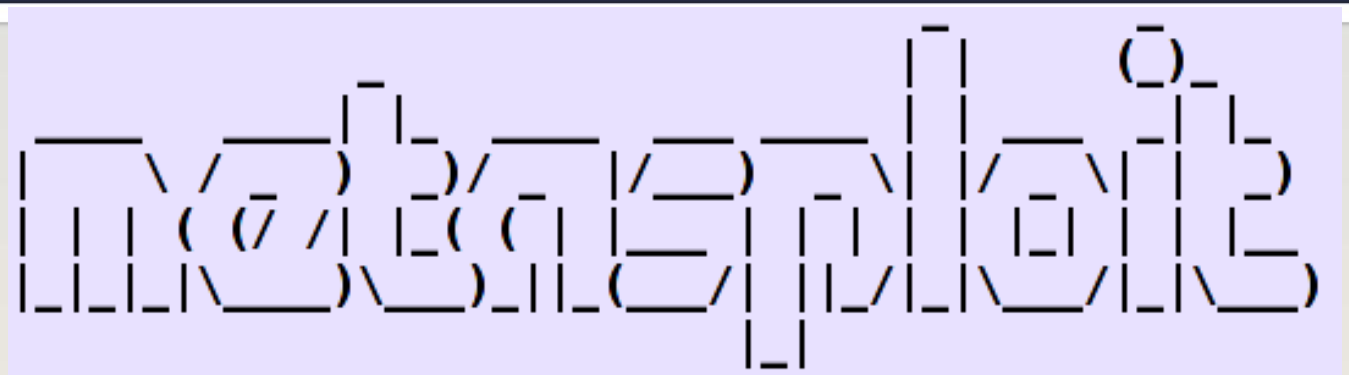
The Metasploit Project

metasploit

http://www.metasploit.com/



The Metasploit Framework



```
       =[ msf v3.3-dev
+ -- --=[ 294 exploits - 124 payloads
+ -- --=[ 17 encoders - 6 nops
       =[ 58 aux
```

metasploit

Interfaces

- Gui
- Web
- Console
- cli

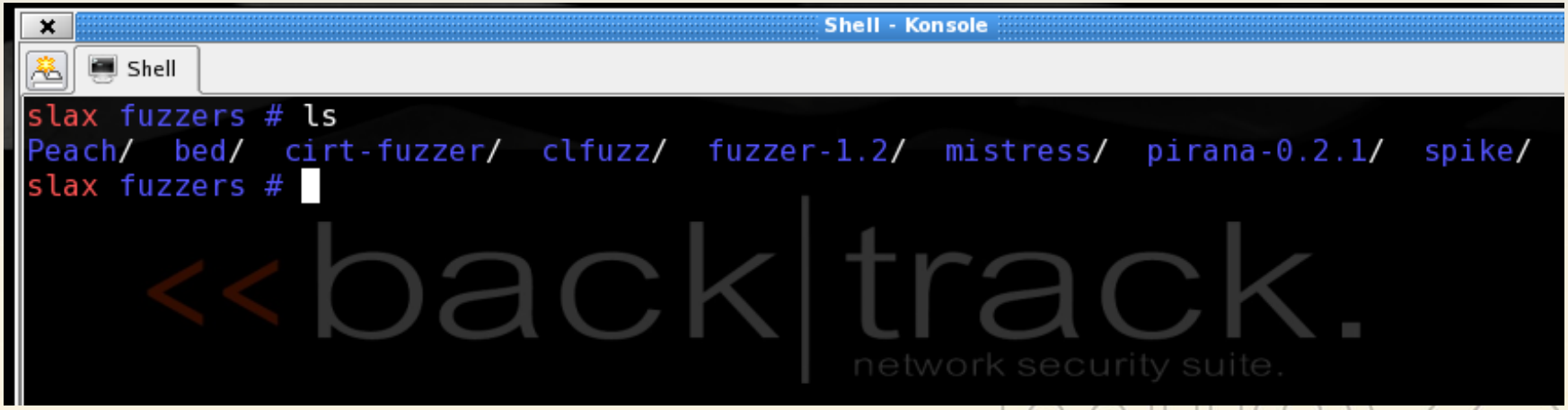
Meterpreter

The purpose of meterpreter scripts are to give end-users an easy interface to write quick scripts that can be run against remote targets *after* successful exploitation. (*Metasploit)*

Meterpreter is an effective tool for creating backdoors.

Fuzzers

A Security fuzzer is a tool used by security professionals (and professional hackers) to test parameters of an application. Typical fuzzers test an application for buffer overflows, error handling, and format string vulnerabilities.



SaintExploit at a Glance

Exploits vulnerabilities found by the SAINT vulnerability scanner.

Allows the user to verify the existence of vulnerabilities by exploiting them and gathering evidence of penetration.

Features exploit tunneling that allows you to run penetration tests from an exploited target. Features seamless integration with SAINT's GUI.

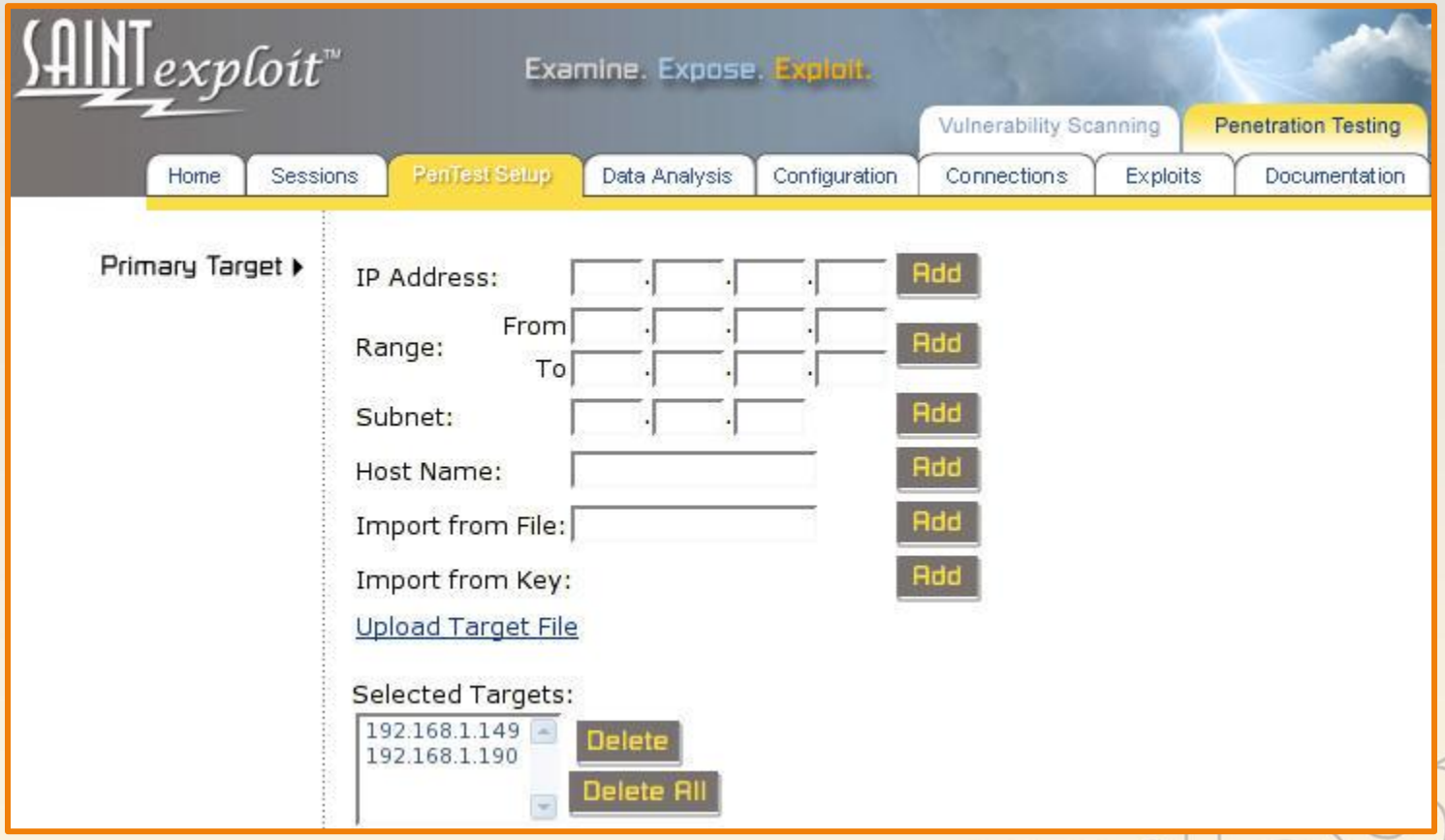
Boasts an extensive, multi-platform exploit library. Includes remote, local, and client exploits.

Provides automatic penetration testing.

Runs individual exploits on demand.

Includes Web site emulator and e-mail forgery tool with built-in design templates.

SaintExploit Interface



Core Impact Overview

Highly Automated Penetration Testing Capabilities

- Commercial-Grade Exploits
- Schedule, One-Step Penetration Testing Capabilities
- Rapid Penetration Testing (RPT)
- Web Application Testing
- Comprehensive Information Gathering
- User Credential Capturing
- Comprehensive 100% Python
- Rich Reporting Capabilities

The steps in the RPT include:

1. **Information Gathering**
2. **Attack and Penetration**
3. **Local Information Gathering**
4. **Privilege Escalation**
5. **Clean Up**
6. **Report Generation**

Core Impact

Summary of vulnerability validation process

Solid Reporting!

Details of vulnerability validation process

Host: /192.168.36.20

Vulnerabilities imported from Retina module:

BID-10108

Vulnerability validation status: MSRPC LSASS Buffer Overflow exploit failed.

BID-10114

Review

How Do Exploits Work?

Lets look at a few types of exploits!

- Format Strings
- Race Conditions
- Heap Spraying
- Heap Overflows
- Buffer Overflows

The Metasploit Project

Saint Exploit – Cost Effective Choice

Core Impact In-depth

Module 10 Lab
Advanced Exploitation Techniques