Malware Cheat Sheet

Packers

Introduction

Packing is a technique originally designed to **compress executables**. These days mostly used for **obfuscation** and anti-debugging. **Malware** usually (>90%) **rely on compression and encryption** to **escape detection** from antivirus software.

A *packer* is the program that **transforms** an *original executable* into *packed data* which can be used by an *unpacking stub* to **reproduce** the original executable. A *packed executable* **combines** the unpacking stub and packed data to produce the original executable at runtime.

Operation

1. Packer consumes an input executable
2. **Transformation**. The executable is transformed, usually via **compression** and encryption
3. **Unpacking stub creation**. The unpacking stub is created, it reverses the transformation, reconstructs the import table and hands over execution to the original executable via a **tail jump**
4. **Combination**. The unpacking stub and executable are combined into a packed executable with the unpacking stub as the entry point

Transformation types

- **Compressors** - compress the executable
- **Crypters** - encrypt the executable
- **Bundlers** - bundles several files
- **Protectors** - protects original code from debugging

Packer types

Monomorphic packers **always produce the same output and binary**. They are trivial to detect and can be statically unpacked.

Polymorphic packers **always produce different output which results in the same binary**. May **hinder signature-based detection** (but can be **possible if** done dynamically), static unpacking is likely not possible. May be produced by adding junk code, in-line random data, generate code etc.

Metamorphic packers **always results in modified binaries**. Much like polymorphic packers.

Atomicity

An *atomic* packer unpacks the **whole original executable at once** and can be unpacked manually.

An *incremental* packer unpacks **a unit of code at a time such** as a basic block, single instruction etc. The entire original executable is never in memory. Impossible to unpack manually.

Packer examples

UPX (Ultimate Packer for eXecutables)

- **Supports a large number of formats of executable files**.
- **Provides compression** at the level of WinZip/zip/gzip, and better
- **Fast in-place unpacking**, which does not require additional memory
- Written in **C + +**
- Extensible
- Uses **GNU**
- Uses compression library NRV, UCL

ASPack

- **Paid** packer of executable **Windows files**.
- Packaging **code**, data and **resources**
- Handles **EXE**, **DLL** and **OCX**
- Quick
- Compression ratio is **40-70%**
- Compatible with C++, Visual Basic, Delphi etc. for Win32

FSG - *F*ast *S*mall *G*ood

- **Free** packer of executable **Windows files**
- **158B** loader
- **TLS support**
- **Written in Assembly**
- Ideal for small applications in Assembly

WinUpack (Ultimate PE Packer)

- **Free** packer for executable **Windows files**
- Compresses resources
- Uses LZMA

Morphine

- Its main use if for **avoiding antivirus** detection
- **Obfuscates unpacking code**
- **Polymorphic engine** with random data
- Uses a PE loader to **complicate memory dumping**
- Open source

- **Allows recursive usage**

Unpacking

One may use **static unpacking** if the packing procedure is known - to extract the binary immediately from the packed executable.

For more advanced packers, one uses **dynamic unpacking** to evaluate the executable until the executable is unpacked in memory. It can then be extracted from the memory using a dump and a import table fixer (PETools and ImportReconstructor).

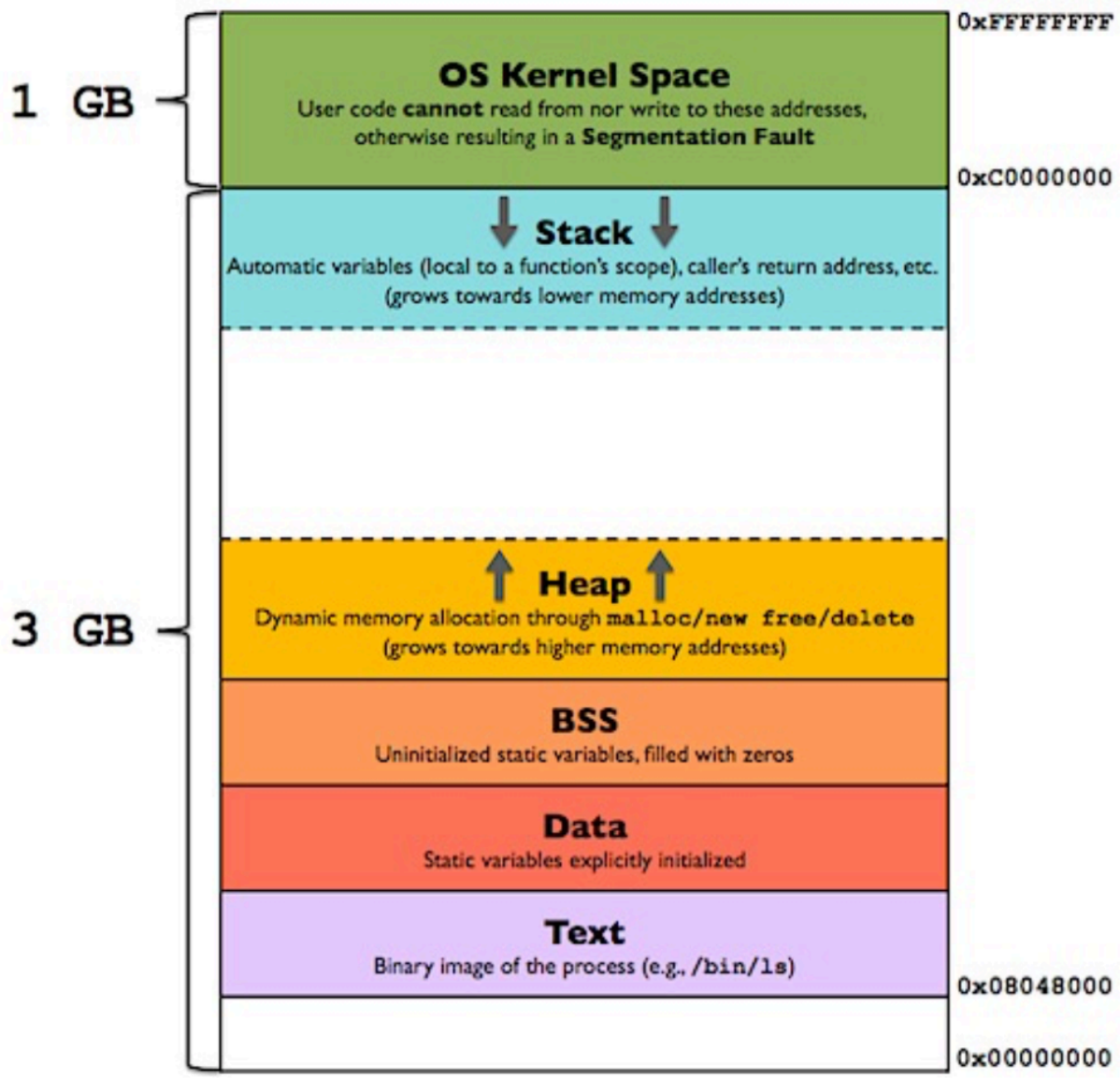
Assembly

Calling conventions

Description	STDCALL	FASTCALL	CDECL
Parameters	Pushed **onto the stack**, **caller** must **clean** up after call	Pushed **onto the stack**, **callee** must **clean** up after call	First two parameters in `ECX` and `EDX`, **rest onto the stack**. **Caller** must **clean** up after call
Return value	EAX	EAX	EAX
Non-volatile registers	EBP, ESP, EBX, ESI, EDI	EBP, ESP, EBX, ESI, EDI	EBP, ESP, EBX, ESI, EDI

Memory layouts

Linux

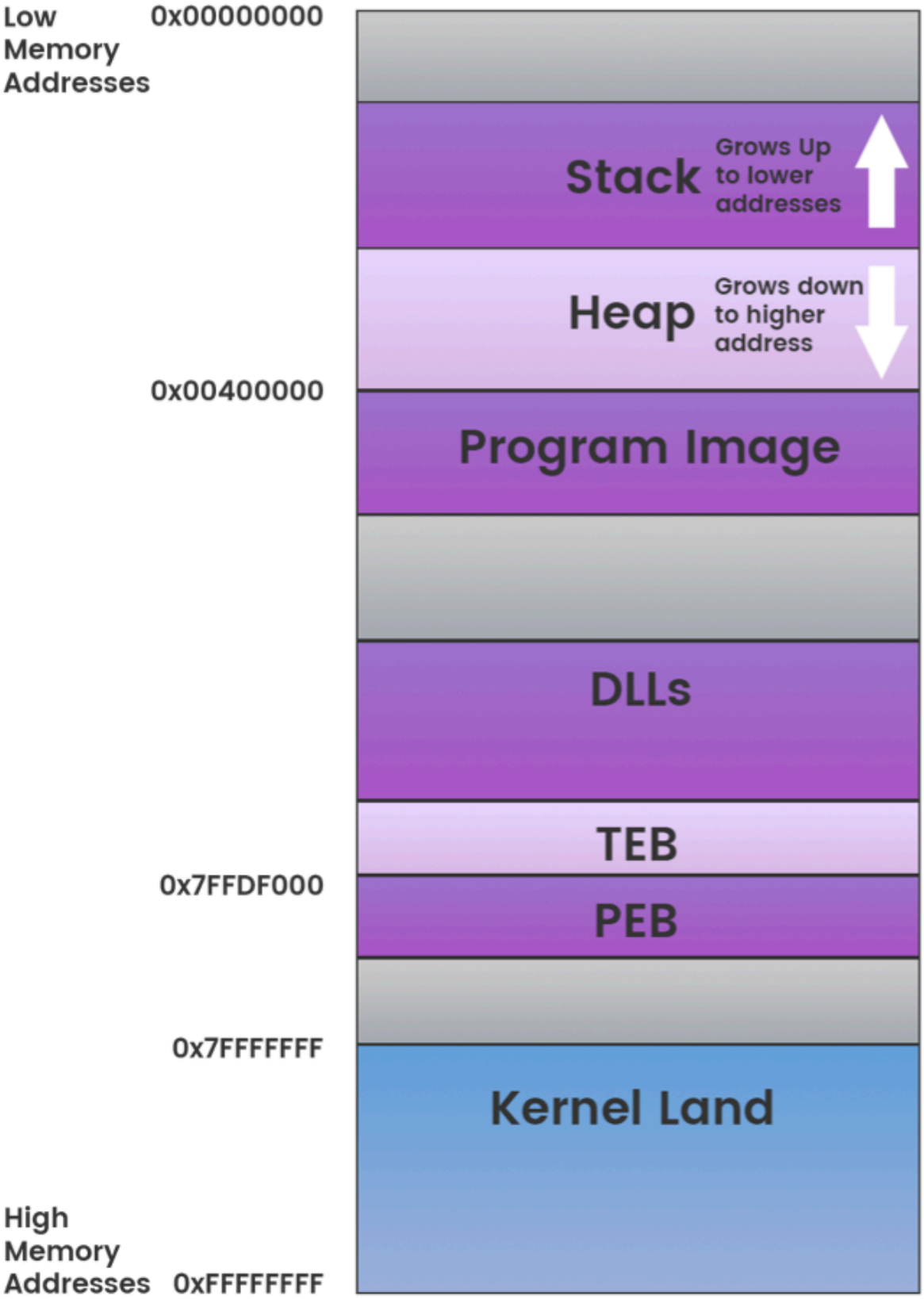


The main four parts of a linux program are:

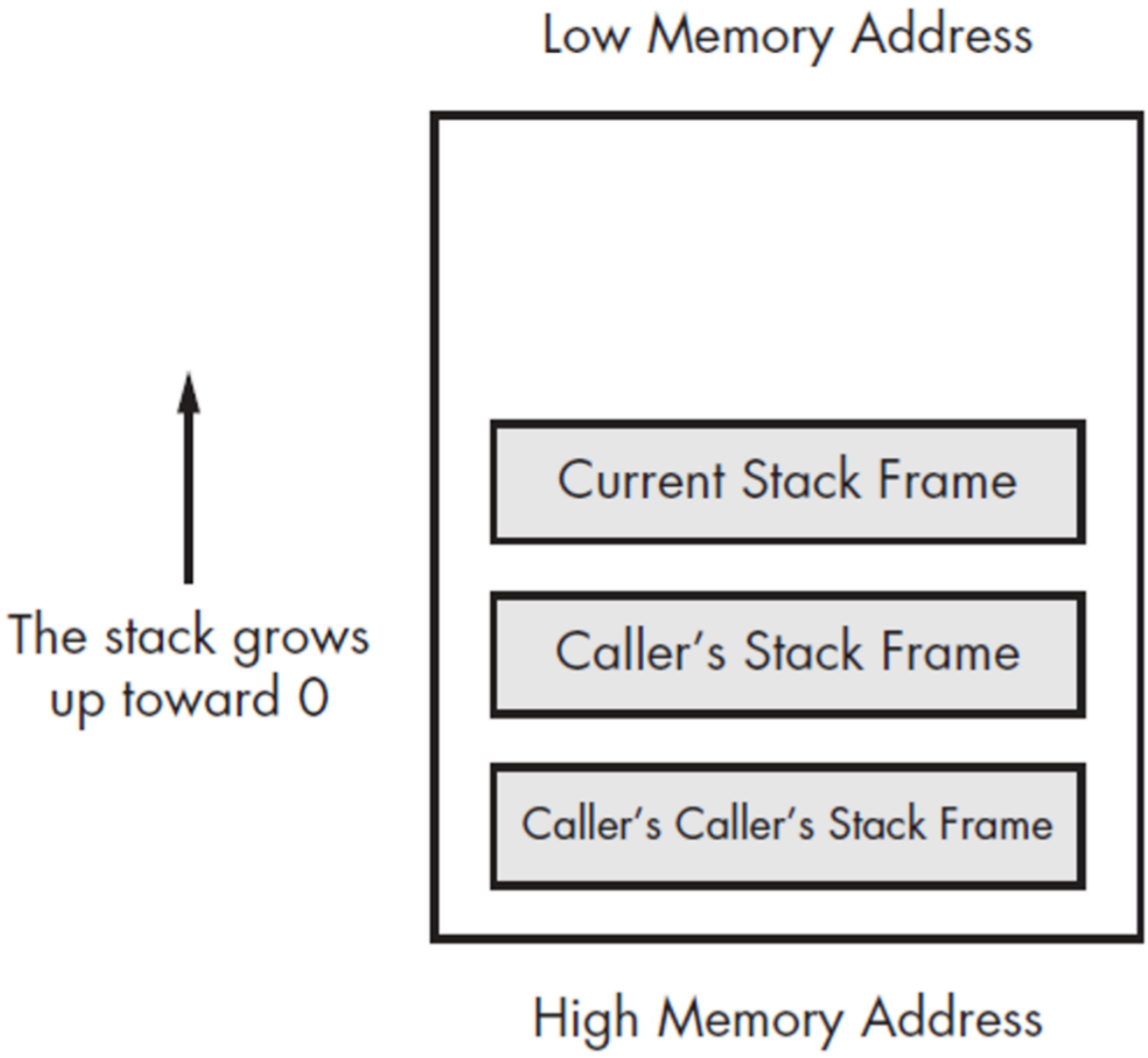
1. **Stack** - "grows downwards" from **high addresses, local scope** etc.
2. **Heap** - "grows upwards" from **lower addresses**, dynamic memory etc.
3. **Data / BSS** - **static variables**, either uninitialized (zeroed) or explicitly initialized
4. **Text** - **binary image of the process**

Windows

WIN32 MEMORY MAP



Stack frames



The **most recent stack frame is at the bottom of the stack**, with its caller before it etc.

Each stack frame contains (in order):

1. **Local variables**... (ESP points to the last one)
2. **Old EBP** (EBP - stack base pointer points here)
3. Return address - points to the instruction after `call`
4. **Arguments**...

Botnets

Introduction

A botnet is a **network of computers** made up of **machines infected** with a malicious backdoor. The controlling server, **command and control center** (C&C, C2, CCC) **communicates** with the bots over a **C&C channel**.

Botnets are used for sending **spam**, performing DDoS attacks, **identity theft** etc.

Types of botnet architecture

- **Centralized** - the C&C communicates **directly with the bots**
- **Decentralized** (P2P, peer-to-peer) - the C&C communicates with **one bot which then spreads the message** to other bots
- **Hybrid** - a **mixture** of the both

Botnet communication channels

- IRC-oriented
- IM-oriented
- Web-oriented
- Other

Lifecycle

1. Botnet **conception** - **designed** and implemented
2. Botnet **recruitment** - infecting machines
3. Botnet **interaction** - registering with C&C
4. Botnet **marketing** - renting, selling source code etc.
5. Attack **execution**

Throughout its life it **may deploy hiding mechanisms** such as **multi-hopping**, **fast-flux proxies**, **ciphering**, **spoofing** etc.

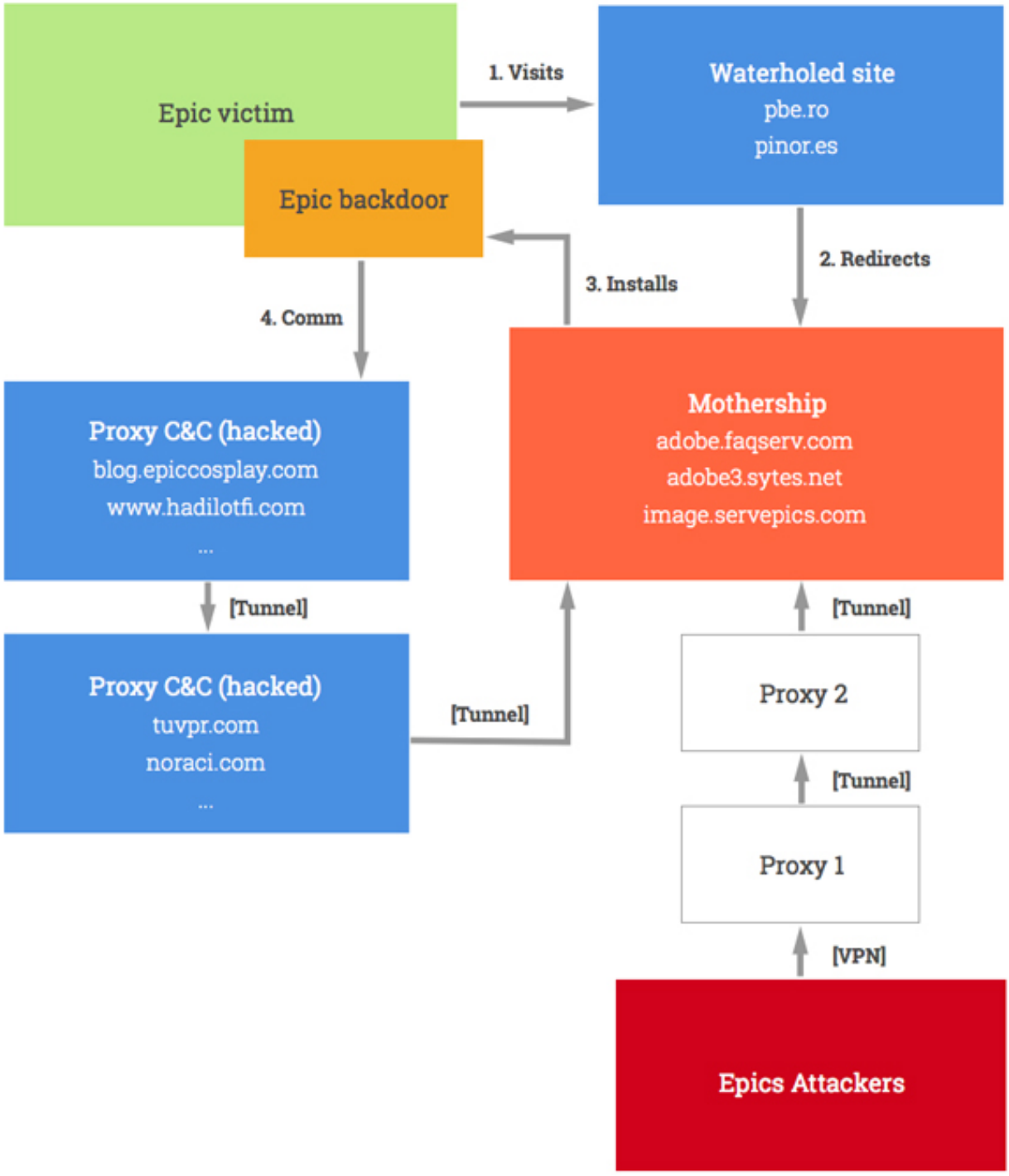
Background radiation

Non-productive network traffic can be seen as **background radiation**. There are different reasons for the radiation:

- **Benign radiation** from **misconfiguration**.
- **Malicious radiation** from flooding **backscatter** - a DoS attack may use **spoofed addresses** and result in the **target responding to requests**, which can detected
- **Malicious radiation** from vulnerability scans

As for **backscatter**, it can be analysed using **darknets**, **honeypots** or both to **observe internet activity** and **malicious scans**.

Epic Turla



Incident response

Digital forensics

Steps

1. **Preparation**
    1. Ensure that there are **dedicated people and tools** to deal with the incident response
2. **Detection / identification**
    1. IDS, firewall, sandbox etc. to **report security issues**
3. **Containment**
    1. **Minimize damage**, isolate compromised devices / VMs, put them on a Security Domain instead of the Internet
4. **Investigation**
    1. **Collect incident-related data**, such as network traffic, files, logs
5. **Analysis**
    1. The incident response team analyzes the data to **report information losses** and indicators of comprimise (**IOCs**).
6. **Remediation**
    1. The incident response team writes a **recommendation for the IT** - what VMs to clean, what keys to exchange etc.
    2. The IT department carries out the recommendations
7. **Prevention**
    1. The incident response team **recommends incident prevention steps**
    2. The IT department carries them out
8. **Lessons learned**
    1. Write an **incident report**
    2. **Analyse** the team's **performance**
    3. Write missing **documentation**

Phishing

Introduction

Phishing is the process of attempting to **acquire sensitive information** by **acting like a trustworthy party**. It's a form of social engineering technique.

Topologies

- **Traditional phishing** uses a **DNS server** to be able to exchange the IP of the hacker's server. The website lifetime is about **60 hours**
- **A rock-phish** topology uses a **DNS server** to resolve a domain to **any number of proxy bots** which in turn communicate with the hacker's server. The website lifetime is about **170 hours**.
- **A fast-flux phishing** topology uses a **DNS server** to resolve a domain to **any number of dynamic pools of proxy bots** which in turn communicate with the hacker's server. The website lifetime is about **200 hours**.

Malware

Types of malware

- **Worms** - **automatic propagation** without human assistence, spreads itself over network channels
- **Viruses** - **human-assisted propagation** such as an email attachment, tries to replicate itself
- **Trojans** - **hides** its malicious operations as a program that performs some desirable function for the user
- **Rootkits** - **modifies the OS** to hide its existence
- **Backdoor** / trapdoor - used for **remote control**
- **Logic bomb** - execute a **malicious** behaviour **once certain conditions are met**, such as when a program is ran in the target environment

Virus phases

1. **Dormant phase** - on victim's device, **waiting to be activated** (not all viruses has this phases)
2. **Propagation phase** - **copies itself** onto other programs or into certain system areas on the disk.
3. **Triggering phase** - It is activated by some event.
4. **Execution phase** - **payload is executed**

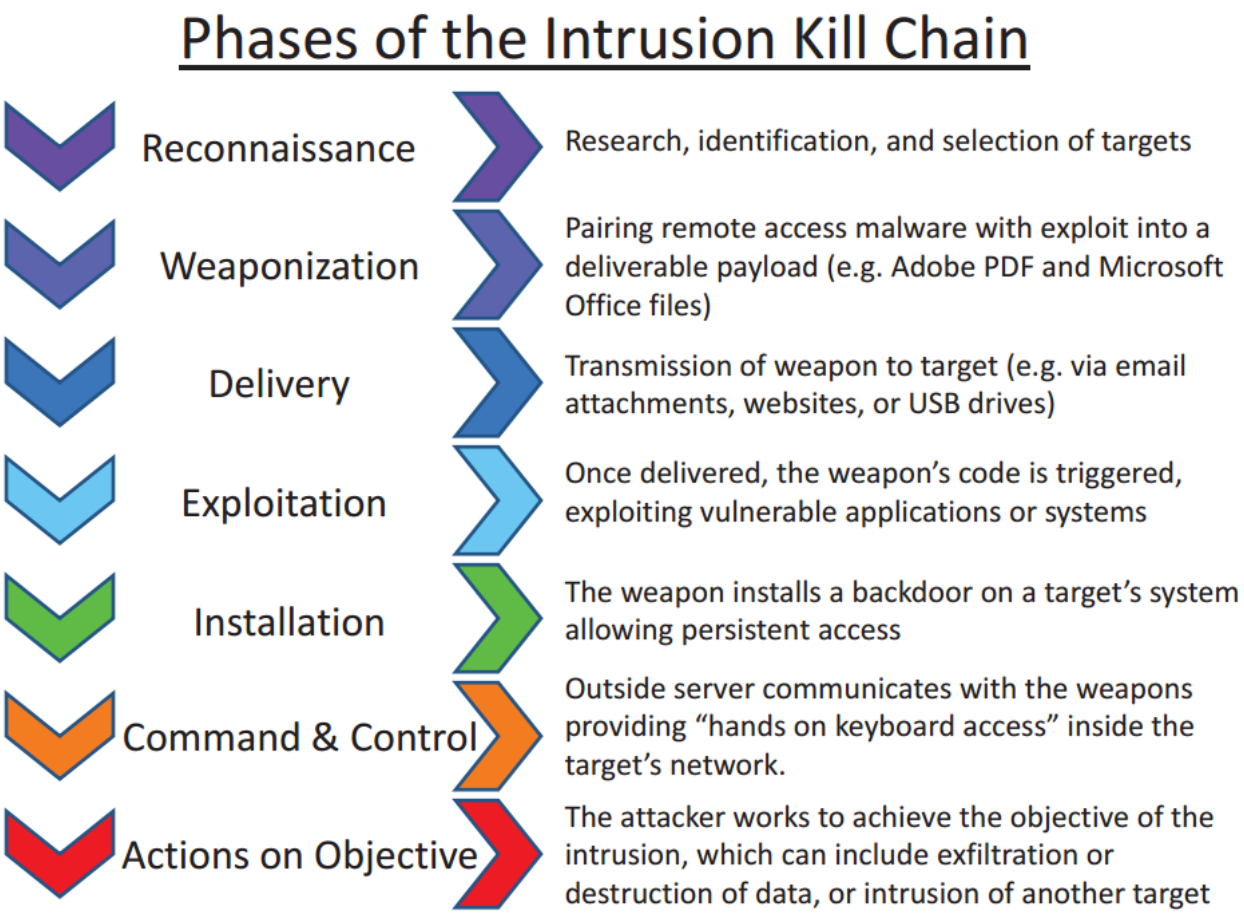
Worm phases

1. **Dormant phase** - on victim's device, **waiting to be activated** (not all worms has this phases)
2. **Propagation phase** - Search for **other systems to infect**, through the network, email etc. And tries infects them.
3. **Triggering phase** - It is activated by some event.
4. **Execution phase** - **payload is executed**

Trojan phases

1. **Penetration phase** - Placing itself into the system.
2. **Activation phase** - The trojan is activated to perform the function for which it was intended.
3. **Execution phase** - The function is performed

Cyber Kill Chain



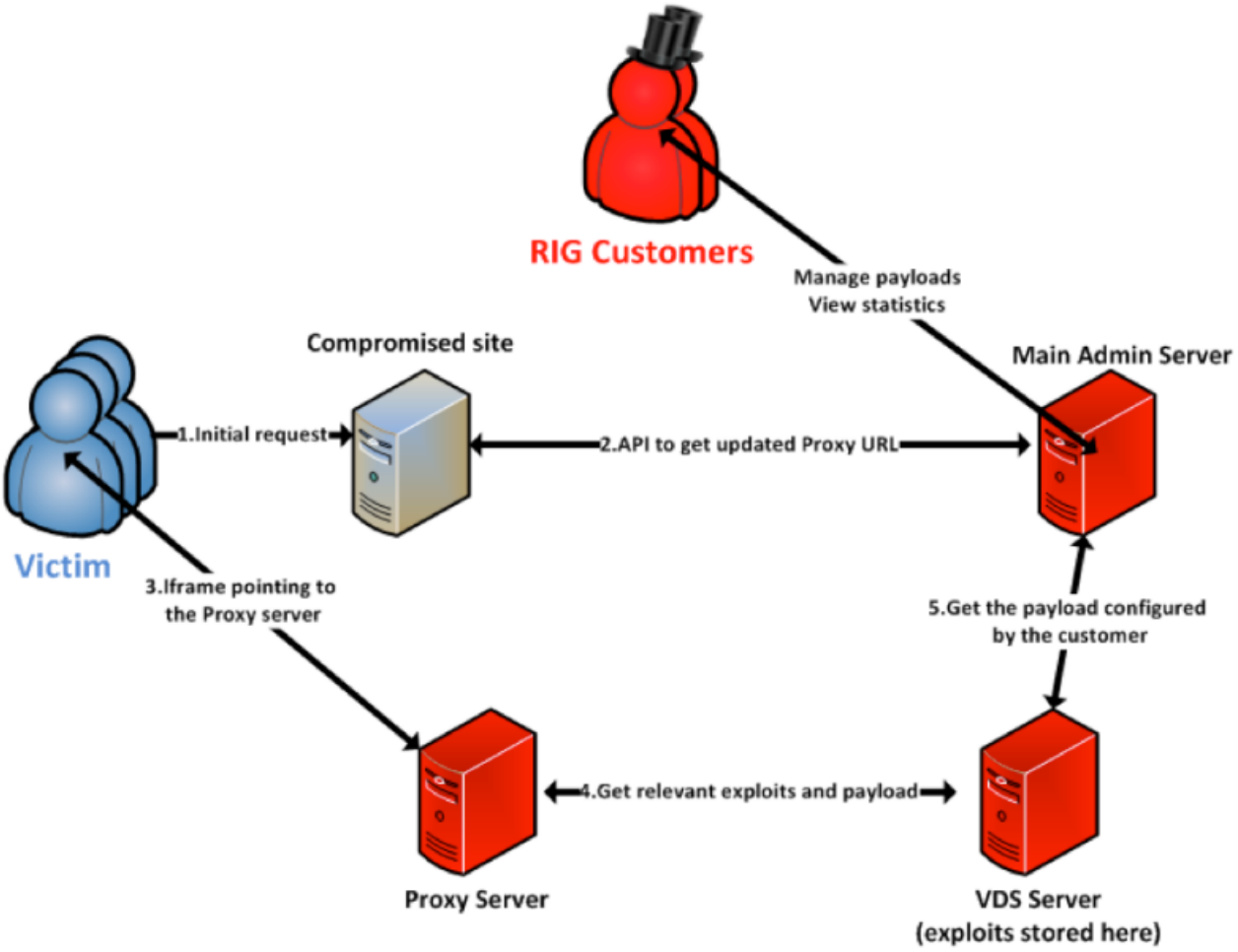
Exploit kits (EKs)

Exploit kits are a prepackaged bundles of tools that can be purchased or leased for the purpose of distributing malware. Popular exploit kits are SweetOrange, Angler, Magnitude, and RIG.

The exploit kit contains all the code to necessary to carry out the following steps:

1. **Establish contact** with the victim through a compromised landing page. And may filterer out victim that meet a certain criteria.
2. **Redirect** to an alternative landing page and detect vulnerabilities in the host that can be exploited.
3. Carry out the **exploit** to spread malware.
4. **Infect** the host environment by executing the malware.

RIG infrastructure



TeslaCrypt (3)

Feature	Description
Delivery method	Landed via a drive-by attack with the help of the Angler web exploit
Platform/ File type	Windows/EXE
Files encryption method	AES-256-CBC using the OpenSSL library code
Session key encryption method	The key is used as a multiplier in the calculated ECDH shared secret sent to the C&C server and stored in a header of encrypted files
Encryption locations	Exceptions: Windows, Program Files, and Application Data. Encrypted in shared folders and removable drives
Deleting backup	Yes, using vssadmin.exe to delete shadow copies of files
Communication with C&C server	URL varies on the build version, data transmitted in an encrypted way (AES-256-CBC) with the hard-coded key
Decryption service	in Tor network
Payment	$500 equivalent in BTC, doubled every 60 hours