# Attack Planner: Systematization and Expansion of Persistence Knowledge

by

Eric Jiang

B.S. Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Howard Shrobe
Principal Research Scientist
Thesis Supervisor

Accepted by . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

# Attack Planner: Systematization and Expansion of Persistence Knowledge

by

Eric Jiang

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

## Abstract

The internet has become a component of society's critical infrastructure. However, the benefit of using the internet has been accompanied by an increasing volume of cyberattacks. Although documentation of these cyberattacks does exist, it is not readily machine processable are often in a form that is even hard for people to understand. In order to protect systems against these attacks, companies have to hire penetration testers to help them find vulnerabilities within the system. However, this can be very expensive and time consuming. It is also very hard to be completely thorough and comprehensive with penetration testing as there are so many different types of attacks.

The AttackPlanner is tool developed at CSAIL that allows users to easily understand the flow of an attack campaign as well as the different ways adversaries can achieve their goals, by representing cyberattacks in the form of trees called *attack trees*. In parallel with the development of the Attack Planner, CALDERA is another tool that assists in this project. My focus of this project is to expand the AttackPlanner's plan repertoire, and its capabilities. There are many different purposes to which cyberattacks are put; this thesis focuses on the persistence aspect of attacks. By persistence, we assume that the attacker already has penetrated the system and can execute a malicious process, but the attacker's goal is to implant an "advanced persistent threat" (APT) that can survive system reboot and continue exploiting the system over sustained periods of time.

Thesis Supervisor: Howard Shrobe
Title: Principal Research Scientist

## Acknowledgments

This thesis is only possible due to the support from the following individuals. It is impossible to show how much I appreciate them. I am forever grateful.

First, I would love to thank my thesis advisor, Dr. Howard Shrobe, for his invaluable advice, patience, and guidance throughout this project. I am truly grateful for being able to work with such a great and knowledgeable person. I will forever cherish the conversations we've had, both technical and non-technical.

I would like to also thank my mother, sister, relatives, and grandparents for the support over the years. Their support has always driven me to succeed and be happy with whatever I do.

Finally, I would like to say thank you to my friends who have been with me throughout my years at MIT. I love you all. I am looking forward to the interactions we have in the next chapters of my life.

After submitting this thesis, I am looking forward to starting the next chapter of my life in industry. Reflecting upon my undergraduate and graduate years, I believe that I am very lucky with the opportunities I have had when it comes to all aspects of my college life. Whether it is regarding my academics, career, or social life, I am forever grateful for all the opportunities MIT has provided for me.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation Background

The internet has become a component of society's critical infrastructure. However, the benefit of using the internet has been accompanied by an increasing volume of cyberattacks. Although documentation of these cyberattacks does exist, it is not readily machine processable and is often in a form that is even hard for people to understand. In order to protect systems against these attacks, companies have to hire penetration testers to help them find vulnerabilities within the system. However, this can be very expensive and time consuming. It is also very hard to be completely thorough and comprehensive with penetration testing as there are so many different types of attacks.

The AttackPlanner is tool developed at CSAIL that allows users to easily understand the flow of an attack campaign as well as the different ways adversaries can achieve their goals, by representing cyberattacks in the form of trees called *attack trees*. In parallel with the development of the Attack Planner, CALDERA is another tool that assists in this project. My focus of this project is to expand the Attack-Planner's plan repertoire, and its capabilities. There are many different purposes to which cyberattacks are put; this thesis focuses on the persistence aspect of attacks. By persistence, we assume that the attacker already has penetrated the system and can execute a malicious process, but the attacker's goal is to implant an "advanced persistent threat" (APT) that can survive system reboot and continue exploiting the system over sustained periods of time.

MITRE has produced the ATT&CK matrix which documents different types of cyberattacks in one centralized framework. However, the descriptions of these attacks are in English, often cryptic and inadequate, and are not amenable to machine processing. Once one is able to understand a particular attack, system managers will have to test these attacks on their own systems to find vulnerabilities. This practice is known as penetration testing (or Pen Testing). Pen Testing is time consuming and requires expertise that is in short supply. After the penetration testing stage, one will also have to figure out ways to patch this vulnerability or find mitigation tactics to prevent exploits from happening. Furthermore, there are a large variety of system types, ranging from cloud systems, hardware servers, IoT devices, electrical grid, nuclear plant, and and other industrial control systems. Given that one instance of pen testing is time consuming, having to do this across many different types of systems introduces a multiplicative time factor that is not scalable.

## 1.2 Overview of Project

As a result, in this project, I have taken steps to reach the goal of creating an automated system that can document these kinds of attacks in an easily understandable manner. I focused on one broad type of attack tactic known as persistence. There are a few existing software systems that are the foundation of this project. The first of which is the AttackPlanner[18] developed by Principal Research Scientist Howard Shrobe. The AttackPlanner is a tool that is able to generate the sequence of steps in an attack campaign, given a particular environment and a system. This tool also visualizes these attacks in a format known as an attack tree which makes it easy to follow when considering the different options an attacker has when executing an attack. The second tool is called CALDERA[9] developed by MITRE. This software is what we have been using to automate simulations of attacks on existing systems.

In Chapter 2, I go into detail about the background and history of research done on attack trees and attack planners.

In Chapter 3, I introduce the AttackPlanner and its capabilities thus far. I also give details on how the AttackPlanner generates plans and the systematization of knowledge used by the AttackPlanner.

In Chapter 4, I introduce the concept of persistence and show how it corresponds in the AttackPlanner. I step through the methods that I've implemented in the AttackPlanner and the model created out of it.

In Chapter 5, I give a summary of my contributions to the AttackPlanner, the use cases of the AttackPlanner, and future works that can be done.

# Chapter 2

# Background Work

## 2.1 Attack Trees

When pen testing was first brought to existence, traditional methods mostly consisted of generating attack trees. An attack tree's root node symbolizes the overarching goal of an attack. Each children of the parent node are equivalent to sub-goals that an attacker needs to achieve in order to achieve the parent node's goal. Some attack trees incorporate the use of logical "and" and "or" gates to show inclusivity and exclusivity of goals. As one goes down the attack tree, the path represents an enumeration of the possible steps in an attack.

One of the first papers that introduced the concept of attack trees was by Bruce Schneier in 1999 as a fault decision tree[15]. Schneier's paper developed the use of fault tree analysis, which analyzes the occurrence of failure in a system. Each parent node in the tree is considered a point of failure in the system, and each child of the parent is a cause of the failure happening. Additionally, each child node has an associated probability of occurring, giving a certain probability of the attack happening in a holistic view. Figure 2-1 is an example of an attack tree from Bruce Schneier's paper.

In Figure 2-1, Schneier outlines the attack tree for opening a safe. In order to open the safe, the attacker would need to do one of the following: pick the lock, learn the combo of the lock, cut open the safe, or the safe would have to be installed improperly beforehand. If the attacker had learnt the combo, then before learning

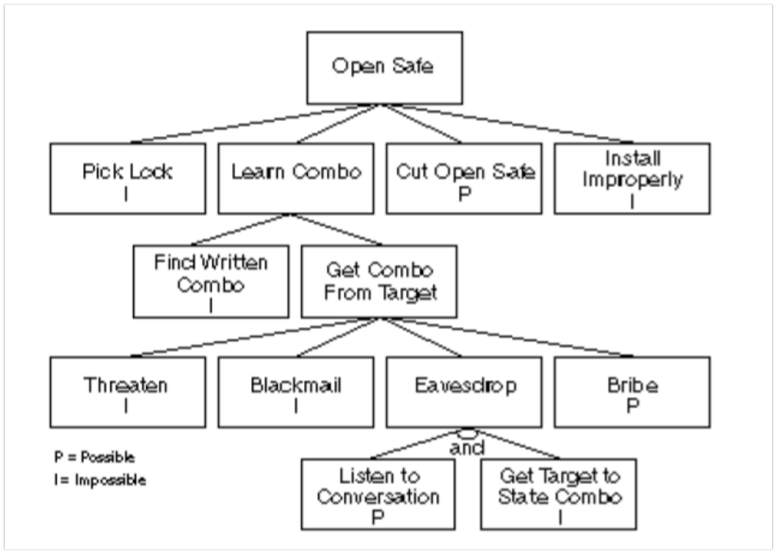


Figure 2-1: Attack Tree Example

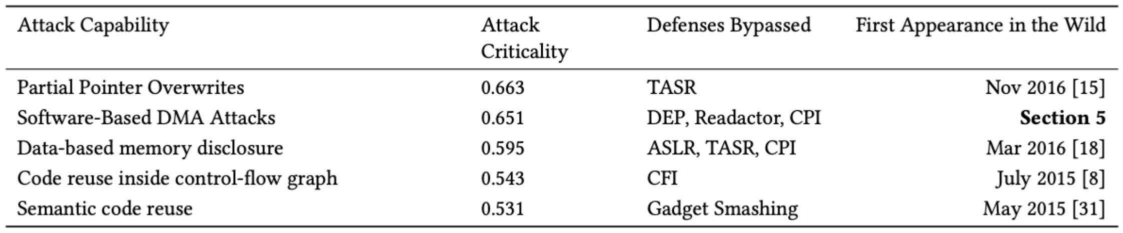
the combo, the attacker would have discovered the combo via finding the combo written somewhere or getting the combo from the victim themselves. The same logic goes for each subsequent level of the tree. Each leaf node is also determined to be either impossible or possible to achieve. For example, "Pick Lock" is deemed to be impossible, denoted by the letter "I" in the node, whereas "Cut Open Safe" is deemed to be possible, denoted by the letter "P". In order to generate this kind of attack tree, one would need to have comprehensive background knowledge of the system, and be able to think from the attacker's perspective.

Risk analysis has been a recent use of attack trees. A paper by Amenaza Technologies Limited, attack trees were viewed as fault trees to give a picture of how much risk was associated with a system and to assess the likelihood of the attack happening[5]. The paper claims that there is a fundamental difference between fault trees and attack trees. The difference is that in fault trees, the leaf nodes are considered to be independent to other leaf nodes. For attack trees, leaf nodes are considered to be interdependent with each other, especially those that are a part of "AND" gates. By combining the concepts of probabilities and leaf independence/interdependence, the paper attempts to model attack trees for risk analysis. With respect to the probabilities associated with each leaf node, they propose establishing a metric or rating for the likelihood of the action to be performed. In addition, they also proposed to attempt to calculate the monetary cost of performing the attack or action.

## 2.2 Attack Planning Predecessors

Since the publication of Schneier's paper, the concept of attack trees has become increasingly popular. With this rise in popularity, comes the creation of attack campaign plans. Attack campaign plans contain a sequence of steps which attackers can use to achieve their goal within an particular enterprise infrastructure. Attack campaign planners are tools that generate these attack campaign plans.

Over the years, several attack planning tools have been developed. Quasar, developed at MIT Lincoln Labs, is an attack planning tool that helps defend against and analyze memory corruption attacks[20]. It helps create attack models that gives the effect attacks can have upon systems, and shows the current coverage of the system's defenses against attackers. The attack trees in this tool were used to represent existing dependencies and defenses in the system that attackers can exploit or bypass to achieve their goals. An example of the defense results of QUASAR are shown below in terms of attack capability and coverage.



| Attack Capability | Attack Criticality | Defenses Bypassed | First Appearance in the Wild |
|---|---|---|---|
| Partial Pointer Overwrites | 0.663 | TASR | Nov 2016 [15] |
| Software-Based DMA Attacks | 0.651 | DEP, Readactor, CPI | **Section 5** |
| Data-based memory disclosure | 0.595 | ASLR, TASR, CPI | Mar 2016 [18] |
| Code reuse inside control-flow graph | 0.543 | CFI | July 2015 [8] |
| Semantic code reuse | 0.531 | Gadget Smashing | May 2015 [31] |

Figure 2-2: QUASAR Linux Getaddrinfo Attack Example

Automating the generation of attack trees has been a goal of several research teams. [21] from the University of Denmark attempts to create a logical formula that characterizes an attack tree. The top level idea of this process is: Given a process and a goal of this process, we can generate a formula by backwards chaining from the goal in the formula. The generation of this formula includes the use of converting a process into a set of formulae in the form of $\phi \rightarrow \hat{p}$ where $\phi$ is the "antecedent" and $\hat{p}$ is the "consequence" of this "antecedent". From these formulae, the authors can derive a recursive back chaining function from the bottom level "consequences" to the top level "antecedent".

Wing, Lippman et al.[16] use state machines to generate attack graphs. First, they model the network as a finite state machine in which state transitions correspond to attacker actions. At the top level, there is a network property noted that the attacker wants to penetrate.

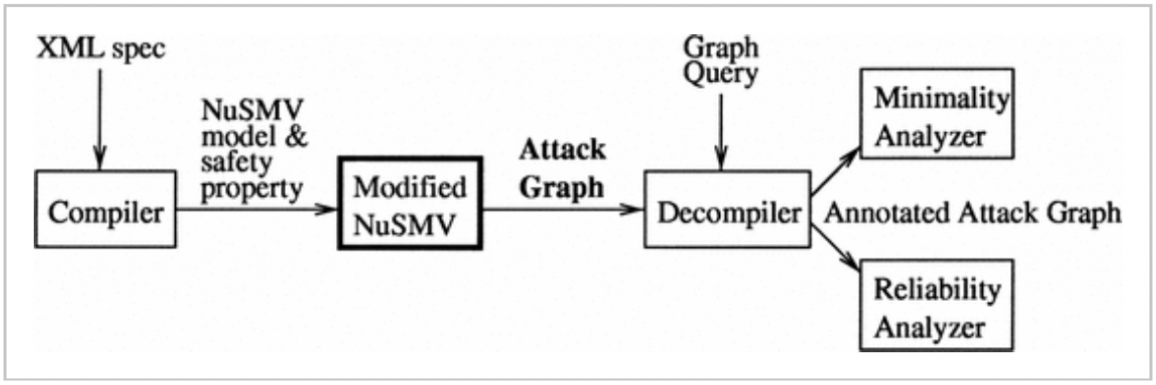


Figure 2-3: Tool suite with NuSMV model checker

The team uses a modified version of the NuSMV model checker to create an attack graph. Then, they analyze the attack graph in two different ways. The first of which is to use the attack graph to figure out what is the minimal number of mitigations necessary to thwart the attacker's plan. The second form of analysis requires an assumption that there is data on the probability of certain network events happening. If the attack graph is annotated with these probabilities, the attack graph can be viewed as a Markov Decision Process (MDP). Using MDP value iteration algorithm, the attacker's optimal path or most probable method of attack can be derived[16].

Wing, along with Oleg Sheyner, also created an attack graph toolkit based on several network specifications and attack models. It was used in conjunction with MITRECorp's Outpost and Lockheed Martin's ANGI, which are two systems that consistently gather network data. Using the data from these two systems, the toolkit is able to create network specifications for a particular network. Along with network specifications, they also used corresponding security specifications, or CVE's , to generate attack graphs and visualize them in a GUI. This process is shown below[22].

A team in Grandata Labs in 2012 developed another tool for attack planning. Their solution included the use of a description environment fed into a planner. The

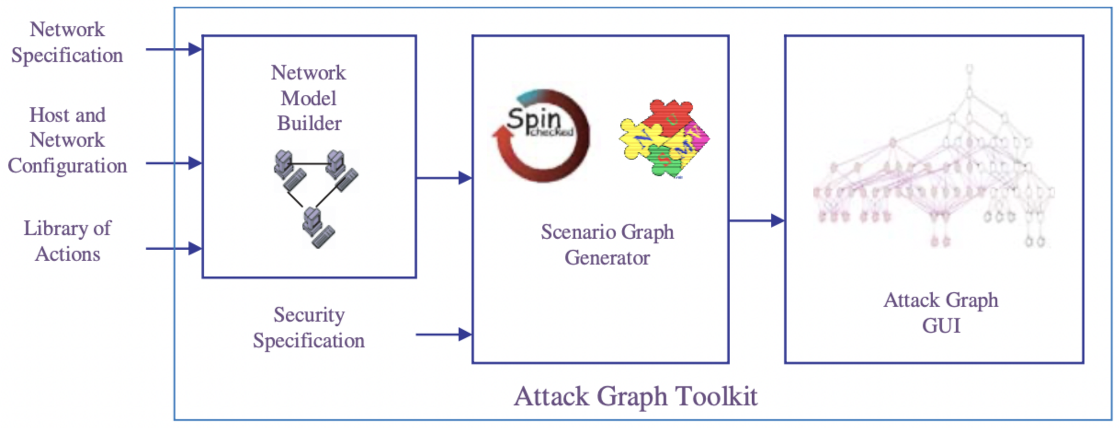


Figure 2-4: Network Attack Graph Toolkit Architecture

planner would then send its attack trees to a penetration testing framework to simulate penetration testing[13].

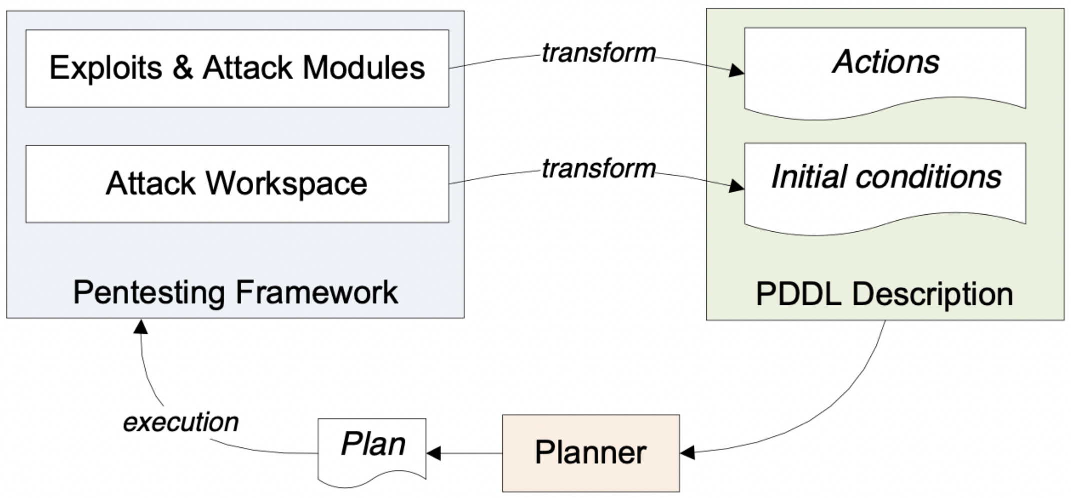


Figure 2-5: Architecture of Grandata Lab's solution

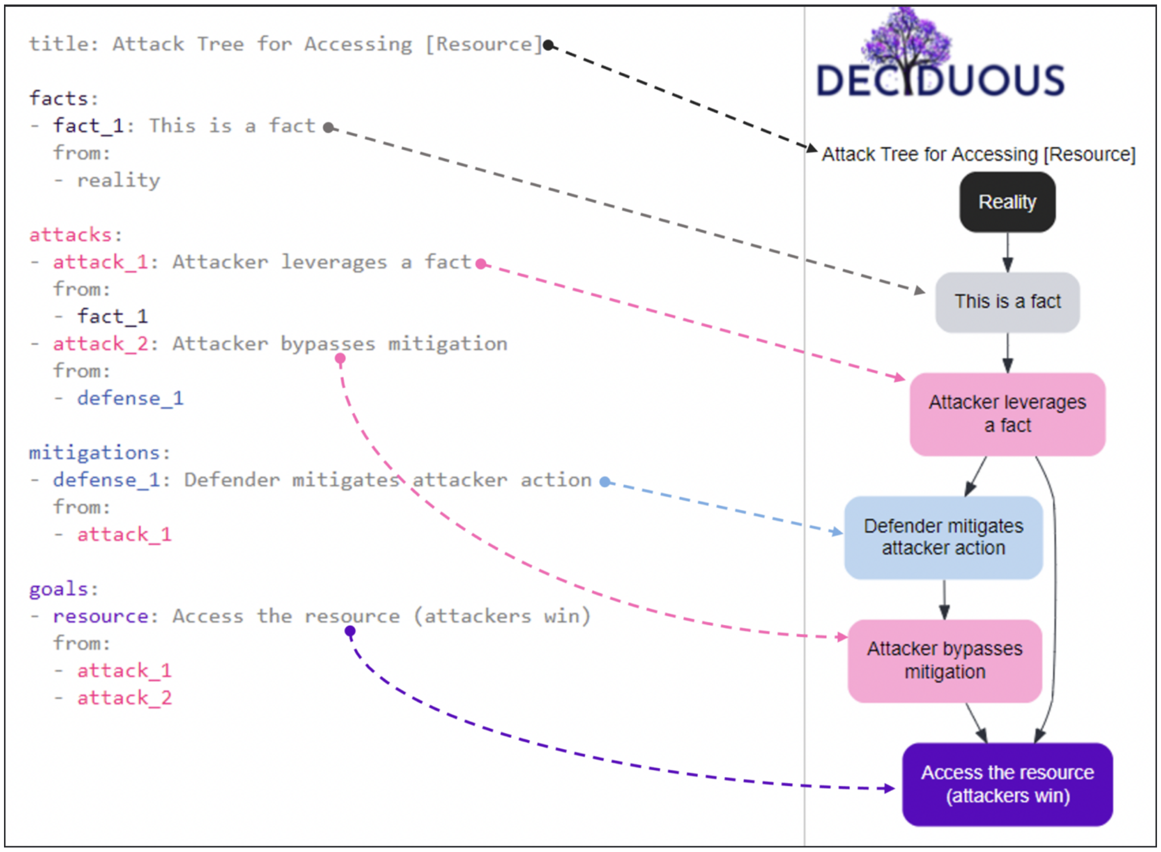
As somewhat of a follow up to this paper, the author published a paper on finding the optimal attack path given an attack tree for non-deterministic environments. The algorithm utilized a modified version of Dijkstra's and Floyd-Warshall's algorithms. They were able to build a probabilistic planner that could solve scenarios with 500 machines. Although the results showed that the run-time of the best path algorithm was slightly worse than a deterministic method used by previous researchers, the team argues that if the implementation had been written in C or C++ instead of Python, it would have been faster[14].

Gregory Falco, a former MIT PHD student, has also conducted research on attack planning, but utilizes it in an interesting context: urban infrastructure. Falco set out to create risk models for SCADA systems, given that IoT devices are prone to a variety of high impact attacks such as worm attacks and propagation attacks. Additionally, Falco talks about the usage of attack planners being able to mitigate, and aid in formulating a defense response against attacks[3].

A team at Princeton University developed a tool called MulVAL, which is an end-to-end framework that executes vulnerability analysis on networks. This tool has shown decent results during its presentation and testing at the Red Hat Linux platform. It was able to detect 84% of the Red Hat vulnerabilities reported in the Open Vulnerability Assessment Language (OVAL), which is a vulnerability language. It includes a reasoning system based off of certain rules that represent exploits, compromise propagation, and multi-hop network access. It also has an analysis algorithm that utilizes a network's policy specification and attack simulations[11].

Another system that was created to utilize attack planners on network security was NetSPA. The overarching goal of NetSPA was to compute attack graphs for user-specified networks. In NetSPA, users were able to define a network and its configuration, create models of actions that attackers could utilize, and generate all possible attack graphs for that network. It utilized several different databases to help generate these attack graphs. The most important database was its action database which contained a collection of actions that attackers could take against a network[1].

In addition to the above research reports, there have also been a few commercial software tools that generate attack trees and attack plans. The first of which is developed by ISOGRAPH called AttackTree. This tool uses attack models to calculate the probability of an attack's success given a system. This tool also allows users to calculate the cost and difficulty of these attacks by manually entering cost values and difficulty ratings for specific actions in the attack trees[6]. Another tool that is not exactly an attack tree generator, but is related is called Deciduous. This tool is mainly used as a security decision tree generator. In this model, there are a few components to the trees. They include: facts, attacks, mitigations, and goals. Facts are concepts that are true to a system. Attacks are specific actions that attackers can take against the system. Mitigations are specific actions that defenders can take against attacks. Goals are the attacker's end goal for an attack[17].



```
title: Attack Tree for Accessing [Resource]

facts:
- fact_1: This is a fact
  from:
  - reality

attacks:
- attack_1: Attacker leverages a fact
  from:
  - fact_1
- attack_2: Attacker bypasses mitigation
  from:
  - defense_1

mitigations:
- defense_1: Defender mitigates attacker action
  from:
  - attack_1

goals:
- resource: Access the resource (attackers win)
  from:
  - attack_1
  - attack_2
```

DECIDUOUS
Attack Tree for Accessing [Resource]
Reality
This is a fact
Attacker leverages a fact
Defender mitigates attacker action
Attacker bypasses mitigation
Access the resource (attackers win)

Figure 2-6: Deciduous Decision Tree Example

As you can see from this image, the decision tree gives possible actions an attacker can take, ways that a defender can mitigate this attack, and an attacker's counterattack to bypass this mitigation in order to achieve the end goal.

## 2.3 CALDERA

CALDERA is the second tool used. Developed and released by MITRE in 2019, it is a comprehensive tool that allows developers to automate and simulate attacks in a closed and safe environment. CALDERA takes an adversary profile as an input and is able to execute an attack plan based on that profile[9]. Additionally, this system was built around MITRE's ATT&CK framework, which is a comprehensive attack matrix of many different types of attacks[10]. This results in CALDERA being proficient in variety as it contains adversary profiles from the three major operating systems: MacOS, Linux, and Windows. Additionally, because CALDERA uses an adversary profile as an input, it seems feasible to feed in the attack tree output from the AttackPlanner as a way to link up the two software systems.

# Chapter 3

# AttackPlanner

## 3.1 Brief Overview

The AttackPlanner is a tool that, given a particular computational environment, is able to generate an attack graph, a set of attack campaigns specifically for that environment [18]. The attack graph encompasses many possible ways in which this goal can be met as there are usually many different ways of going about achieving this goal. As a result, a path on the tree represents an attack plan for the goal at the root node. This makes it easy to understand for people who are viewing the details of the attack tree.

Because the AttackPlanner is given a computational environment, the Attack-Planner needs to have comprehensive knowledge of the victim. As a result, the AttackPlanner is able to model the various systems and hardware out in the world such as operating systems, network topology, file systems, and so on. Another feature of the AttackPlanner is that it's able to model the dependencies of the computational resources of the environment, such as data integrity, user capabilities, and privacy.

The base knowledge of the AttackPlanner comes from reasoning about the steps within an attack campaign, while also taking into account the environment this attack campaign is executed in, as well as the goal that is achieved. As a result, systematization of knowledge for the AttackPlanner is key to its enhancement.

## 3.2 Generating an Attack Campaign

When generating an attack campaign in the AttackPlanner the first step is to identify a top level goal. Then the AttackPlanner identifies the underlying sub-goals needed to achieve that top level goal and then creates sub-trees to achieve the sub-goals. This reduction to sub-goals is repeated until the sub-goal is simple enough to be achieved via a subset of primitive actions defined in the AttackPlanner. The leaf nodes in the attack plans are actions instead of sub-goals. This is known as hierarchical task net planning, which works by problem reduction rather than generative state space search[19]. In this case, we are trying to find smaller sub-goals with each iteration level.

The generation of sub-trees in the attack planner is done by an algorithm called Depth First Backtracking Search. This algorithm was first popularized by the language Prolog, which is one of the first logic programming language developed and still used in AI development[2].

The AttackPlanner is primarily written in a domain specific language (DSL) embedded in Common Lisp. The style of this DSL is similar to the Planning Domain Definition Language (PDDL), which is a standard encoding language for planning tasks[4]. The features of this language include the use of a few concepts. The first of which is the use of objects, which are the things in the world that are related to the planning task. The second of which are predicates. Predicates represent the properties with respect to the objects in relation to the environment or planning task. The third is the initial state, which is a description of the environment before the attack campaign begins.. The fourth is the overarching goal of the planning task, known as the goal specification. Lastly are the actions that are used to change the state of the environment. This DSL is itself implemented in the Joshua reasoning system (itself a DSL based in Common Lisp). Joshua provides a database of assertions (also called predications) as well as forward and backward chaining rules that draw inferences from the set of assertions.

## 3.3 Defining the Enterprise Threat Environment

The AttackPlanner needs two pieces of information in order to compile an attack campaign. They are the structure of the victim enterprise and the servers that the attacker controls. The AttackPlanner uses a set of macros to define these structures. These macros expand into code that create assertions that set the foundations of the Allegro Common Lisp (Joshua) reasoning system. Joshua is the language used for the AttackPlanner because it contains features that make planning easier to implement. It does this by allowing the change of data structures without having to change the knowledge structure of the original data structure, and is able to incorporate other tools without modifying the tool[12]. The macros defined in the AttackPlanner allow the attacker to look for and understand the enterprise structure given the elements the attacker already has. Some key macros are detailed in Table 3.1.

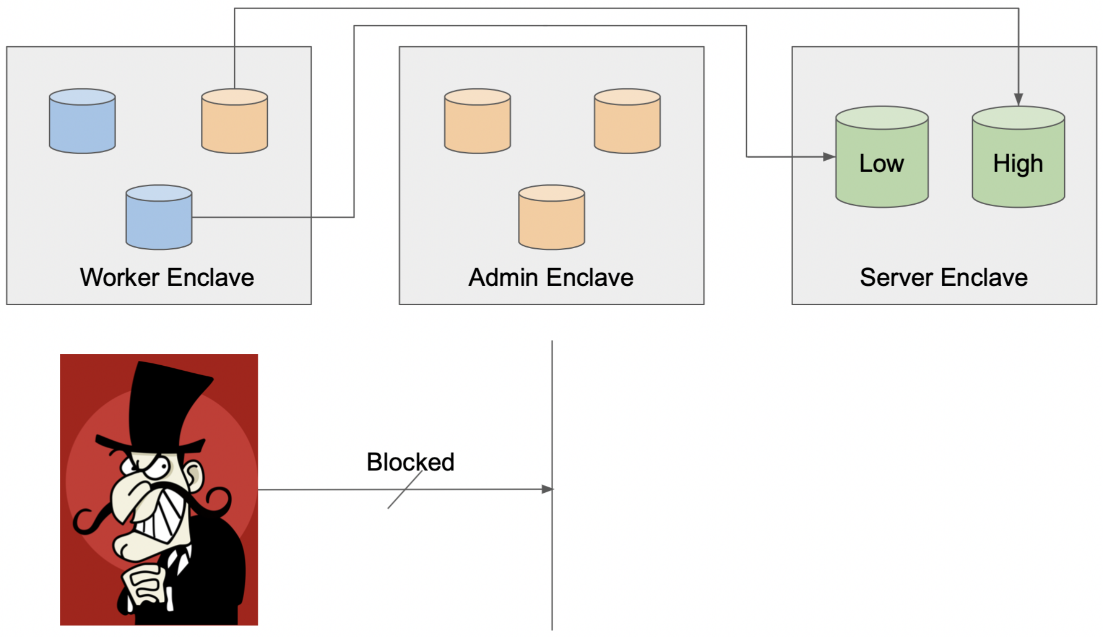


Figure 3-1: Wilee Example Graphic

To demonstrate these macros for better understanding, we can set up an example environment. We will use the "Wilee" example in the AttackPlanner as an example. Figure 3-1 shows a general overview of the Wilee example. The top level goal of the

| Macro | Definition |
|---|---|
| defexternal-internet | an external attacker's internet IP address |
| define-attacker | the certain properties of an attacker |
| define-enterprise | the victim entity that the attacker is trying to affect |
| defresource | the objects or resources that are available in the environment, whether it is for the attacker or the victim |
| defauthorization-pool | defining a group of identities and privileges that are managed in a unified way |
| defcapability | creating a privilege or permissions that exists in the topology |
| defsite | defining enclaves or domains of an entity |
| defsubnet | defining sub-domains of an enclave or a larger domain |
| defensemble | defining a group of computers that are uniform |
| defcomputer | creates a computer entity in the environment |
| defuser | creates a user or entity |
| defblacklist | defines what entities are not allowed in the network |
| defprocess | defines a process that is going to be executed in the environment |
| defrouter | creates a router that belongs to an entity |
| defswitch | creates a switch associated with an entity |

Table 3.1: Select Macros in AttackPlanner

Wilee example is for the attacker to corrupt the high database server in the server enclave. However, the attacker is blocked from accessing it. As a result, the attacker has to work around this by utilizing the worker enclave. The issue is that the worker enclave only has access to the low database server in the server enclave. Only admins have access to the high database server. Thus, one method to get access to the high database server is to steal the admin credentials. One way to do so is to somehow bait the admin to log into one of these worker enclave machines using the admin's credentials, which allows the attacker to steal the credentials. We now have the attack plan of the Wilee example as follows:

1. Log into a worker machine

2. Install two pieces of malware on this machine
    - Key logger malware
    - Kill disk malware
3. Lure an admin to log into the compromised machine
    - Fill the disk with nonsense
    - Alert the admin about abnormal activity
4. Use key logger malware to steal admin credentials
5. Use admin credentials to log into high database server
6. Attacker now has control of the high database server

Now that we have the attack, it's time to translate the environment into the AttackPlanner. We define the attacker IP address that is outside the victim environment via *defexternal-internet*. An example is:

```
(defexternal-internet outside ("192.168.0.0" "255.255.0.0"))
```

Besides the attacker IP address, we can also define or assume traits of the attacker via *define-attacker*. We can say that the attacker lives in a location outside of the victim environment, and possesses a download and adware server.

```
(define-attacker attacker
    :location outside
    :download-servers attacker-download-server
    :adware-servers attacker-adware-server)
```

Next comes the definition of the victim enterprise. The victim enterprise needs to be thoroughly detailed due to having an accurate picture of the environment. The AttackPlanner needs to know:

1. The network topology

2. Computers and computing resources

3. Users using the computers to access the computing resources

4. The access control matrix

We can define enterprise as follows:

```
(define-enterprise victim)
(defauthorization-pool victim-authorization-pool)
(defcapability sysadmin victim-authorization-pool)
(defcapability user-read victim-authorization-pool
    :greater (sysadmin))
(defcapability user-write victim-authorization-pool
    :lesser (user-read)
    :greater (sysadmin))
```

Here, we see that the enterprise name is "victim". Additionally, the privileges of the victim are defined by the victim-authorization-pool. Lastly, the capabilities or permissions of the victim enterprise's members are defined by *defcapability*. This is how the permissions of the access control matrix is defined. They are defined in a lattice structure as you can see. First, we have the highest privilege level which is indicated by sysadmin having the victim authorization pool. The sysadmin has the privilege to user-read and user-write indicated by the keyword *greater*. The lesser keyword indicates that the current privilege is higher than the corresponding privilege. Thus, user-read is a strictly less privilege than user-write.

Next, we define the access control matrix's computing resource via *defresource*. The high database server has a high data server as the computer, and in order to access this resource one must have the following permissions:

- Write to the high data server

- Read the contents of the high data server

This is how the high database server is defined:

```
(defresource high-database database
    :computers (high-data-server)
    :capability-requirements ((write data-high-write)
                              (read data-high-read)))
```

We also have to define the users of the worker enclave. This includes the computing resources of the workers and the permissions.

```
(defuser typical-worker-bee
    :user-type normal-user
    :ensemble worker-computers
    :computers (typical-worker-computer)
    :typical t
    :capabilities (user-write)
    :authorization-pools (victim-authorization-pool)
    :has-weak-password 'yes)
```

One note for this particular macros is that it represents the entirety of the user space, indicated by *:typical t*. This reduces the redundancy of having to manually add individual users to the AttackPlanner and allows for a user abstraction.

We can also define a group of worker computers via *defensemble* as follows:

```
(defensemble worker-computers
    :enterprise victim
    :size 40
    :address-range ("192.168.0.0" "255.255.255.0"))
```

This shows that the worker computers are part of the victim enterprise, the number of worker computers there are, and the IP addresses of these computers.

Within this ensemble, we can define a typical computer as:

```
(defcomputer typical-worker-computer windows-computer
    :ip-address-string "192.168.0.3"
```

```
        :typical t
        :authorization-pool victim-authorization-pool
        :ensemble worker-computers
        :superuser ())
```

This states that the typical worker computer has the IP address of 192.168.0.3, is representative of a typical computer, is part of the victim-authorization-pool, and is part of the worker-computers ensemble.

The last few important macros are *defrouter*, *defswitch*, and *defblacklist*. *defrouter* and *defswitch* define routers and switches that act as gateways to the servers. In order to go from one machine to another, they will have to pass through a router and the switch associated with the destination.

```
    (defrouter victim-router ("192.168.0.1"  "192.168.10.1"
                              "192.168.20.1")
        :authorization-pool victim-authorization-pool
        :superuser typical-sysadmin
        :external-networks (outside))
```

Here we have the victim router with the specified IP addresses. It contains the victim-authorization-pool and only the sysadmin is able to access it.

```
    (defswitch worker-net-switch wired-switch "192.168.0.2"
   :authorization-pool victim-authorization-pool
   :superuser typical-sysadmin)
```

The switch defined is the worker network switch. As a result, if anyone were to enter the worker network, they would have to pass through the victim router and this worker-net-switch.

Lastly, *defblacklist* define the "firewall" rules of the network topology. Each router and switch will have their own rules of what to forward and what not to forward. Everything that is defined by *defblacklist* will not be forwarded.

```
(defblacklist (telnet victim-router)
    :block everywhere)
```

This states that any telnet packet will be rejected by the victim-router. This includes telnet packets that originate from the IP address of the victim environment.

## 3.4 AttackPlanner Keywords

Outside of the low level threat enterprise description, there is also the attack methods that exist in the attacker's disposal. These attack methods comprise of key words that define actions, goals, steps to the attack, and in some cases, prerequisites for the attack to happen. Object definitions and typings are also declared in these attack methods. Table 3.2 has a detailed list of the important the keywords for attack methods, actions, and goal definitions.

| Keyword | Definition |
|---|---|
| defattack-method | defining an attack method |
| define-action | defining the actions that are available to attackers |
| define-goal | defining the goals that an attacker can achieve |
| to-achieve | the top level goal for the attacker to achieve |
| output-variables | the "return" value, which can be information about what has been compromised |
| bindings | defining objects |
| guards | rules that need to be satisfied |
| typing | defining object types |
| prerequisites | the conditions for the attack method to take place |
| plan | the steps to the attack method |
| sequential | a modifier to the plan that indicates each step are done in order |
| goal | sub-goals of the attack method |
| actions | actions involved in the attack method |
| post-conditions | the change that has happened after the attack method has executed |
| attack-identifier | the attack method's ID according to MITRE's ATT&CK matrix |

Table 3.2: Select Keywords in AttackPlanner

## 3.5 Using Macros and Keywords

To show these macros and keywords in the AttackPlanner, we describe a lateral motion attack. When it comes to accomplishing an attack, usually the attacker is unable to directly take control of the victim server. To circumvent this, the attacker will typically get a foothold of a machine that is connected to the main victim target and achieve remote execution upon that machine. Because the attacker can remotely execution from the foothold, the attacker can then affect the main target. This process is detailed in Figure 3-2.

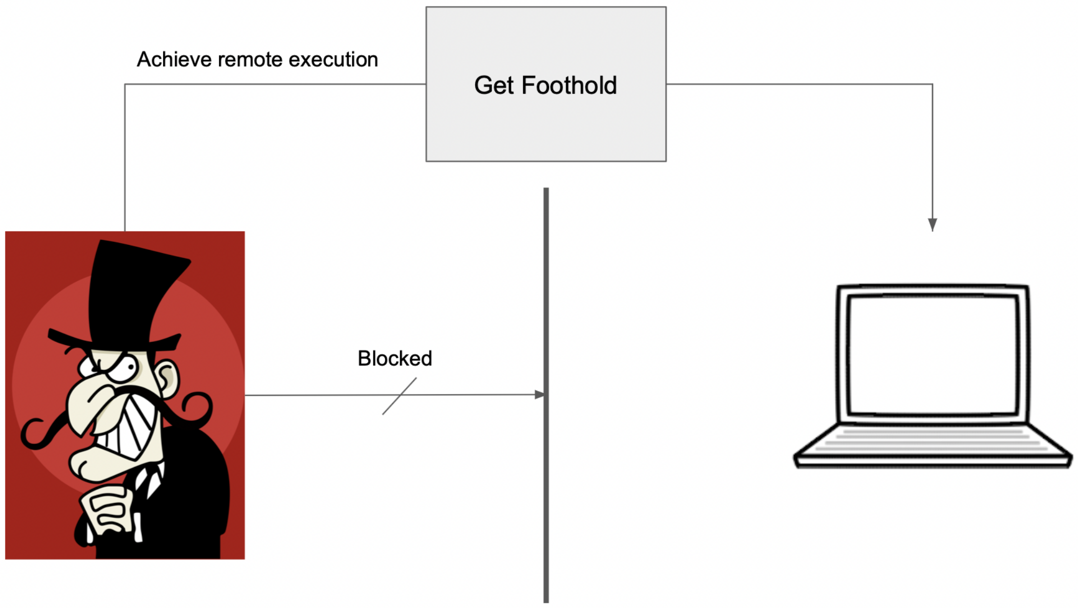


Figure 3-2: Lateral Motion Graphic

To see this graphic translated to the AttackPlanner, we have Figure 3-3.

First, we have the *defattack-method* keyword to declare the attack method called *lateral-motion*. Next we have the top level goal of *to-achieve* which is to get a foothold on a machine that is related to the target machine. Then, there is the *bindings* and *typing* blocks that instantiates any objects or entities that describe the attacker's resources and the enterprise threat environment. *guards* is similar to the concept of prerequisites, but includes the *not* keyword that negates the rules. The main portion of the attack method lies in the *plan* keyword in which the steps of the attack method

```
(defattack-method lateral-motion
  :to-achieve [get-foothold ?victim-computer ?protocol-name]
  :bindings (;; (?victim-os ?victim-computer.os)
              ;; Now find somebody that can make the connection, accepts connection will find one if there is one
              [accepts-connection ?victim-computer ?protocol-name ?new-foothold-computer]
              [current-foothold ?current-foothold-computer ?]
              [attacker-and-computer ?attacker ?])
  :guards ([not [place-already-visited? ?victim-computer foothold nil]]
            [foothold-doesnt-exist ?victim-computer]
            ;; Use this method only if you can't get a connection to the victim from where you are
            [not [accepts-connection ?victim-computer ?protocol-name ?current-foothold-computer]]
            ;; Owned computers are computers that the attacker controls other than
            ;; his primary machine.  No point in trying to move to them since
            ;; they have no more ability to get to the victim than the attacker's
            ;; primary machine
            (not (member ?new-foothold-computer (owned-computers ?attacker)))
            )
  :typing (;; (?victim-os operating-system)
            (?victim-computer computer)
            (?current-foothold-computer computer))
  :plan (:sequential
            ;; Make a note that we've already considered this place as a foothold to
            ;; prevent looping back to here while trying to achieve remote execution
          (:note [place-visited ?victim-computer foothold nil])
          ;; Now see if the attacker can gain remote execution on the new-foothold-computer and in what role
          ;; (?new-foothold-role is a return value)
          (:trace "trying for remote execution on ~a from ~a" ?new-foothold-computer ?current-foothold-computer)
          (:goal [achieve-remote-execution ?new-foothold-computer ?new-foothold-role])
          ;;If so then actually make the connection to the victim from the new foothold
          ;; (:goal [make-connection ?victim-os-instance ?protocol-name ?remote-execution-state ?output-contet])
          (:action [connect-via ?new-foothold-computer ?new-foothold-role ?victim-computer ?protocol-name])
          (:trace "connected to foothold ~a ~a ~a ~a" ?new-foothold-computer ?new-foothold-role ?victim-computer ?protocol-name)
          )
  :post-conditions ([has-foothold ?victim-computer ?new-foothold-computer ?new-foothold-role ?protocol-name])
  )
```

Figure 3-3: Lateral Motion Method

are enumerated. We have a sequential plan as follows:

1. Check to see if the current machine has been used as a foothold before. If it was used before, then find another machine. Otherwise, keep using.

2. Achieve remote execution on the foothold machine.

3. Connect the foothold computer to the target computer.

Lastly, the final result once this attack has succeeded is that the attacker has a foothold on the target machine.

```
(define-action connect-via (?current-foothold-computer ?current-foothold-role ?victim-computer ?protocol-name)
  :ignore (?current-foothold-role)
  :prerequisites ([accepts-connection ?victim-computer ?protocol-name ?current-foothold-computer]
                  [is-protocol ?protocol-name])
  :post-conditions ([connection-established ?current-foothold-computer ?victim-computer ?protocol-name])
  )
```

Figure 3-4: Connect Via Action

```
(define-goal achieve-remote-execution (victim-computer victim-role) :outputs (victim-role))
```

Figure 3-5: Achieve Remote Execution Goal

Figure 3-4 elaborates upon how the action *connect-via* is defined and Figure 3-5 shows the goal *achieve-remote-execution*.

## 3.6 Analyzing Function Call Trace

Here is a partial trace outputted from the AttackPlanner regarding the lateral motion attack method:

```
> Trying backward rule LATERAL-MOTION (Goal... )[ACHIEVE-GOAL [GET-
    FOOTHOLD #<OBJECT (HIGH-DATA-SERVER)> DATABASE-PROTOCOL]


> Trying backward rule REMOTE-EXECUTION-TO-CORRUPT-ATTACHMENT (Goal... )[
    ACHIEVE-GOAL [ACHIEVE-REMOTE-EXECUTION #<OBJECT (TYPICAL-WORKER-
    COMPUTER)> ?NEW-FOOTHOLD-ROLE]


> Trying backward rule REMOTE-EXECUTION-VIA-CORRUPT-EMAIL (Goal... )[
    ACHIEVE-GOAL [GET-USER-TO-CLICK-ON #<OBJECT (ATTACKER)> #<OBJECT (
    TYPICAL-SYSADMIN)> ?ANONYMOUS4554 ?ANONYMOUS4555]
> Exiting backward rule REMOTE-EXECUTION-VIA-CORRUPT-EMAIL


> Trying backward rule REMOTE-EXECUTION-VIA-CORRUPT-EMAIL (Goal... )[
    ACHIEVE-GOAL [GET-USER-TO-CLICK-ON #<OBJECT (ATTACKER)> #<OBJECT
(TYPICAL-WORKER-BEE)> ?ANONYMOUS4554 ?ANONYMOUS4555]


> Trying backward rule LATERAL-MOTION (Goal... )[ACHIEVE-GOAL [GET-
    FOOTHOLD #<OBJECT (EMAIL-SERVER)> SMTP]
> Exiting backward rule LATERAL-MOTION


> Succeeding backward rule REMOTE-EXECUTION-VIA-CORRUPT-EMAIL
> Succeeding backward rule REMOTE-EXECUTION-TO-CORRUPT-ATTACHMENT
> Succeeding backward rule LATERAL-MOTION
> Trying backward rule MODIFY-THROUGH-ACCESS-RIGHTS (Goal... )[ACHIEVE-
    GOAL [MODIFY DATA-INTEGRITY #<OBJECT (HIGH-DATABASE)>]
> Trying backward rule ACHIEVE-A-RIGHT-YOU-ALREADY-HAVE (Goal... )[
    ACHIEVE-GOAL [ACHIEVE-ACCESS-RIGHT WRITE #<OBJECT (HIGH-DATABASE)> ?
    OTHER-ROLE]
```

```
> Succeeding backward rule ACHIEVE-A-RIGHT-YOU-ALREADY-HAVE
> Succeeding backward rule MODIFY-THROUGH-ACCESS-RIGHTS
```

We can see that the AttackPlanner is trying to execute a lateral motion attack and checking to see if the rules have been satisfied. Each line in the trace shows the method name to execute along with the arguments provided to the method name. We can also see in the third line, the AttackPlanner tries to send a corrupt email to a sysadmin. However, this fails as indicated by the next line when the AttackPlanner exits the remote execution via corrupt email rule. In the line after, the AttackPlanner tries to send a corrupt email to a worker machine. This method succeeds as indicated by a later line that states "Succeeding backward rule REMOTE-EXECUTION-VIA-CORRUPT-EMAIL".

This is an example of the AttackPlanner's Depth First Search Backtracking. We first visit each node, in this case rules, of the plan. If there is a failure in the node or rule, then the AttackPlanner outputs a fail message. After failing, the AttackPlanner will backtrack to the parent node, and check to see if any other children exist. If the children do exist, then it will go down that branch and try the rules there. Otherwise, the AttackPlanner will backtrack again and the recursion continues until all rules have been attempted.

# Chapter 4

# Persistence

## 4.1 What is Persistence

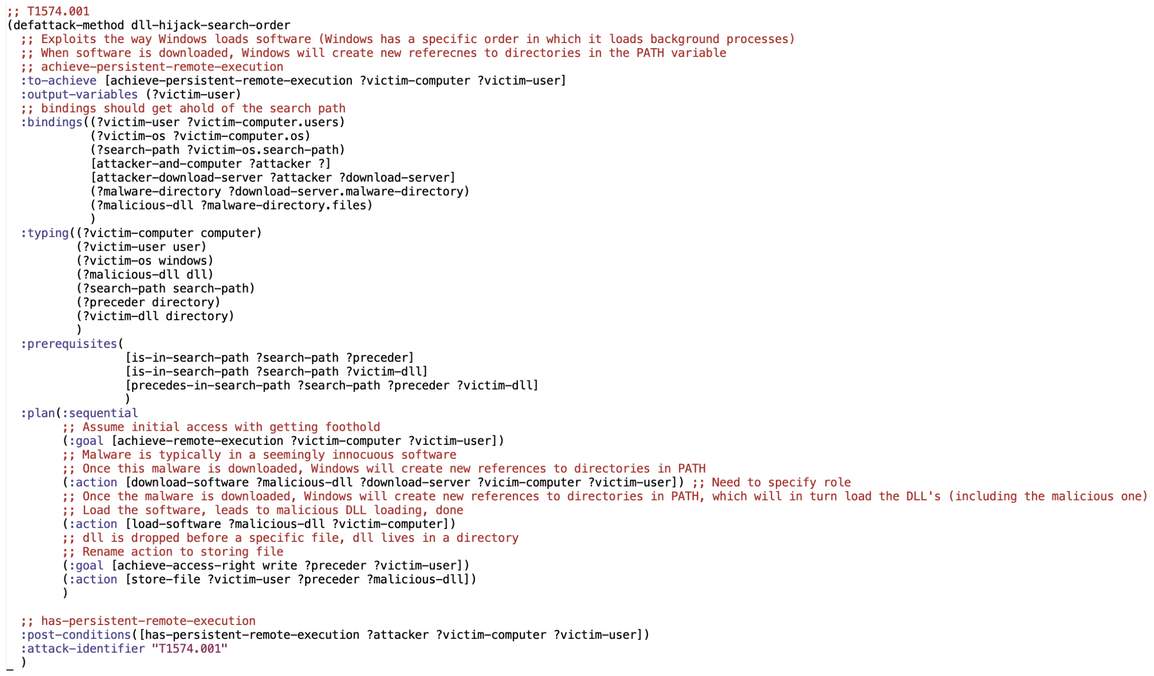
According to MITRE, persistence tactics are attacks that gives the adversary a foothold within a system even if the victim tries to cut off access via restarts, changed credentials, and more. To maintain a foothold on these systems, attackers will generally try to execute access, action, or configuration changes. Some typical methods that allow attackers to execute this are by hijacking the code base or adding startup code[8].

## 4.2 Persistence in the AttackPlanner

In this chapter, we will be talking about the different persistence methods implemented in the AttackPlanner. I assume that initial penetration has already occurred in the system already. The reason being is that initial penetration is a whole other tactic categorized by MITRE. Another reason is for the simplicity of understanding how attacks works. Initial penetration is already a large goal for an attacker, and so is persistence. Generally, in a persistence attack, the attacker will already have some foothold on the system. A few initial penetration attacks are outlined in the AttackPlanner as well.

## 4.3 DLL Hijack Search Order

Systematization of knowledge in the AttackPlanner is necessary since it is the main basis of how the AttackPlanner develops its knowledge. As a result, part of my work involves adding persistence methods in the AttackPlanner.



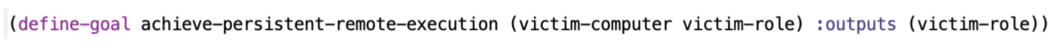
```
;; T1574.001
(defattack-method dll-hijack-search-order
  ;; Exploits the way Windows loads software (Windows has a specific order in which it loads background processes)
  ;; When software is downloaded, Windows will create new referecnes to directories in the PATH variable
  ;; achieve-persistent-remote-execution
  :to-achieve [achieve-persistent-remote-execution ?victim-computer ?victim-user]
  :output-variables (?victim-user)
  ;; bindings should get ahold of the search path
  :bindings((?victim-user ?victim-computer.users)
            (?victim-os ?victim-computer.os)
            (?search-path ?victim-os.search-path)
            [attacker-and-computer ?attacker ?]
            [attacker-download-server ?attacker ?download-server]
            (?malware-directory ?download-server.malware-directory)
            (?malicious-dll ?malware-directory.files)
            )
  :typing((?victim-computer computer)
          (?victim-user user)
          (?victim-os windows)
          (?malicious-dll dll)
          (?search-path search-path)
          (?preceder directory)
          (?victim-dll directory)
          )
  :prerequisites(
                 [is-in-search-path ?search-path ?preceder]
                 [is-in-search-path ?search-path ?victim-dll]
                 [precedes-in-search-path ?search-path ?preceder ?victim-dll]
                 )
  :plan(:sequential
        ;; Assume initial access with getting foothold
        (:goal [achieve-remote-execution ?victim-computer ?victim-user])
        ;; Malware is typically in a seemingly innocuous software
        ;; Once this malware is downloaded, Windows will create new references to directories in PATH
        (:action [download-software ?malicious-dll ?download-server ?vicim-computer ?victim-user]) ;; Need to specify role
        ;; Once the malware is downloaded, Windows will create new references to directories in PATH, which will in turn load the DLL's (including the malicious one)
        ;; Load the software, leads to malicious DLL loading, done
        (:action [load-software ?malicious-dll ?victim-computer])
        ;; dll is dropped before a specific file, dll lives in a directory
        ;; Rename action to storing file
        (:goal [achieve-access-right write ?preceder ?victim-user])
        (:action [store-file ?victim-user ?preceder ?malicious-dll])
        )

  ;; has-persistent-remote-execution
  :post-conditions([has-persistent-remote-execution ?attacker ?victim-computer ?victim-user])
  :attack-identifier "T1574.001"
  )
```

Figure 4-1: DLL Hijack Persistence Attack Method

Figure 4.1 is an example of a persistence attack method called a DLL hijack search order. To give some background, a DLL, on Windows machines, is a Dynamic Link Library that is used by one or more applications or services. When these applications or services utilize a DLL, the DLL will start and run in the background to help them run. Additionally, there can be multiple versions of this DLL file in one system. Usually, an application that has a dependency on a DLL will specify the full path of the DLL that it uses. If no path is specified, then the application or service will find the DLL by scanning the system in a specified hierarchical order called a search order[7]. A DLL hijack search order is when an attacker infiltrates a system and places a malicious DLL in a specified directory of the victim's machine. The malicious DLL has the same name as a legitimate DLL, but is placed in a directory that is earlier in the search order than where the legitimate DLL is located. If it's earlier in the search order, then the malicious DLL will be used instead of the legitimate DLL, thus the word "hijack" in the attack name.

You can see the keywords of Table 3.2 being used in this attack method. The name of this attack method indicated by defattack-method is dll-hijack-search-order. Then there is the top level goal of the attacker achieving persistent remote execution. The to-achieve keyword is also how variables are passed down in the attack method. The output or return variable of this function is the victim user's information, which is to show us who has been compromised. The bindings define the objects used in the method. For example, attacker-download-server is a given object in the AttackPlanner in which we store attacker and download-server information. Then, the malware-directory variable is initialized to the download-server's malware directory. And lastly, the malicious-dll variable comes from the download-server's malware directory's files. The typing contains all the types of the objects being used in this attack. The prerequisites of this attack include a check to see if the directory that the attacker is trying to place a file in is before the legitimate DLL's directory, and a check to see if both DLL's are in the search order. Next, we have the attack-method plan, which is the sequential order of steps to execute this attack. As you can see, the steps include a goal of achieving remote execution (initial penetration), an action of downloading a malicious DLL on the victim computer from the malicious download server, another action of loading this malicious DLL onto the victim computer, a goal of achieving an access write to let the attacker write the file into the system, and the final action of storing this file into the system. The post-condition of this attack is that the attacker now has persistent remote execution upon the victim machine. Finally, the attack-identifier of this attack is T1574.001 on MITRE's ATT&CK matrix.



```
(define-goal achieve-persistent-remote-execution (victim-computer victim-role) :outputs (victim-role))
```

Figure 4-2: Achieve Persistent Remote Execution Goal

In Figure 4-2, I show how a goal is defined in the context of DLL hijack search order. As you can see, victim-computer and victim-role are used and passed down the attack method.

```
(define-aplan-predicate is-in-search-path (path directory) (non-stateful-predicate-model))
(define-aplan-predicate precedes-in-search-path (path before after) (non-stateful-predicate-model))
```

Figure 4-3: Search Path Predicates

In Figure 4-3, I show two predicates that are used in the prerequisites block of the attack method. Predicates are essentially macros that model the network description for generating attacks. The keyword define-aplan-predicate is how one would define a predicate in the AttackPlanner. As you can see, the predicates is-in-search-path and precedes-in-search-path are predicates that help the AttackPlanner identify the system description. This is because is-in-search-path checks to see if a directory is in the search path, and precedes-in-search-path checks to see if a directory is located earlier in the search path than another directory.

```
(define-action store-file (?actor ?directory ?file)
  :bindings ()
  :prerequisites ([has-permission ?actor write ?directory])
  :typing ((?directory directory)
           (?actor user)
           (?file file))
  :post-conditions ([value-of (?directory files) ?file])
  )
```

Figure 4-4: Store File Action

Figure 4-4 shows an action called store-file being made in the AttackPlanner. The keyword define-action creates the action. In order for the action to succeed, it requires three parameters that are indicated by actor, directory, and file. In order for the action to execute, there's also a prerequisite of the predicate has-permission that needs to be fulfilled. Once the action executes, the file that was passed into the store-file action is now located in the directory specified.

## 4.4 Boot or Logon Autostart Execution: Registry Run Keys

One type of persistence techniques involve automatically starting upon a boot up of the machine or logging into a machine. In order to do so, attackers will change system settings or system configurations. Once these changes have taken place and the machine boots up, a persistent program will execute to achieve a secondary goal of privilege escalation. The problem now becomes: How do attackers link or correlate this persistent program to automatically execute upon machine boot up.

One of the ways include manipulating registry run keys. A registry on Windows is a hierarchical database that contains information about configuration settings for Windows applications. Within the registry, there are run keys that indicate what is run upon a machine boot up.

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
```

Figure 4-5: Run Keys that are run upon boot up

Figure 4-5 shows the Run Keys on Windows that execute upon machine boot up. Adding run keys to the registry is quite simple. An attacker can do this remotely via the victim's terminal, which is easy once an attacker gets a foothold on the victim machine. Additionally, the attacker will have to download the malicious persistent payload onto the victim machine. The creation of the run key will have to match and correlate to this malicious file.

Here is the attack method detailing the attack described. There is the overarching goal of achieving remote persistent execution. The definition of the malicious file is located in bindings. The declaration of the registry and the run keys are in typings. Executing this attack is a 3 step process detailed earlier. There is achieving remote execution which is the initial penetration and getting a foothold on the victim ma-

```
(defattack-method boot-or-logon-autostart-execution-registry-runkeys
  :to-achieve [achieve-persistent-remote-execution ?victim-computer ?victim-user]
  :output-variables (?victim-user)
  :bindings ((?victim-user ?victim-computer.users)
             (?victim-os ?victim-computer.os)
             [attacker-and-computer ?attacker ?attacker-computer]
             [attacker-download-server ?attacker ?download-server]
             (?malware-directory ?download-server.malware-directory)
             (?malicious-file ?malwre-directory.files)
             )
  :typing ((?victim-computer computer)
           (?attacker-computer computer)
           (?victim-user user)
           (?victim-os windows)
           (?malicious-file file)
           (?new-key string)
           (?new-value string)
           (?registry registry)
           (?new-entry key-value-pair)
           )
  :plan (:sequential
         (:goal [achieve-remote-execution ?victim-computer ?victim-user])
         ;; action to put run-key in registry, need to link a malicious file to the run key, file can be stored anywhere
         (:goal [install-malware ?attacker ?attacker-computer ?victim-computer ?malicious-file])
         (:action [make-registry-entry ?new-key ?new-value ?registry ?new-entry]))
  :post-conditions ([has-persistent-remote-execution ?attacker ?victim-computer ?victim-user])
  :attack-identifier "T1547.001"
  )
```

Figure 4-6: Registry Run Key Attack Method

chine. The goal of installing malware is the installation of the malicious persistent file. Finally, the registry entry is made with an action detailed in Figure 4-7.

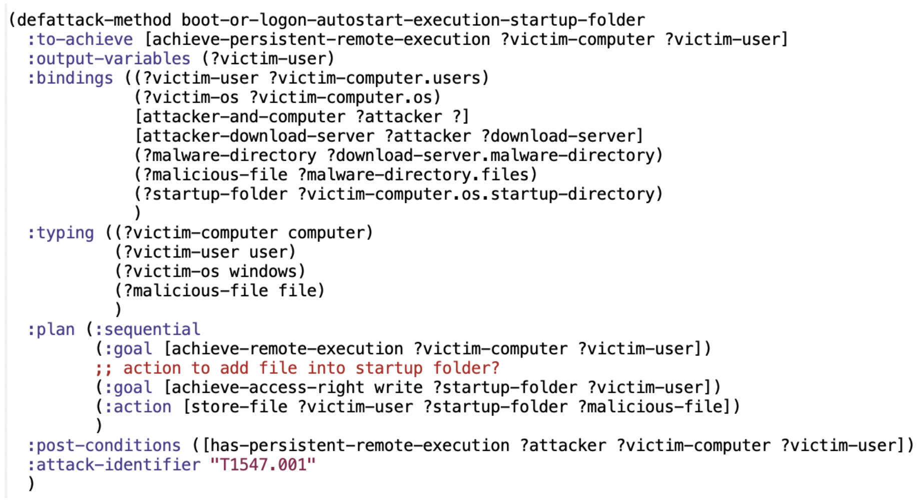
```
(define-action make-registry-entry (?new-key ?new-value ?registry ?new-entry)
  :bindings ()
  :prerequisites ()
  :typing ((?new-entry key-value-pair)
           (?new-key string)
           (?new-value string)
           (?registry registry)
           )
  :post-conditions ([value-of (?key-value-pair key) ?new-key]
                    [value-of (?key-value-pair value) ?new-value]
                    [value-of (?registry entries) ?new-entry])
  )
```

Figure 4-7: Make Registry Entry Action

## 4.5 Boot or Logon Autostart Execution: Startup Folder

Related to the registry run keys is the startup folder. The difference is that when Windows is booted up, the registry run keys are executed first over the startup folder. However, the startup folder does contain information and resources for Windows that are automatically executed upon Window's startup.

In this particular attack, it's very similar to the registry run key attack. The difference is that the malicious persistent file is stored in the Window's startup folder. In order to do so, the attacker will need to have administrative privileges to write a file and store it inside the startup folder.



```
(defattack-method boot-or-logon-autostart-execution-startup-folder
  :to-achieve [achieve-persistent-remote-execution ?victim-computer ?victim-user]
  :output-variables (?victim-user)
  :bindings ((?victim-user ?victim-computer.users)
             (?victim-os ?victim-computer.os)
             [attacker-and-computer ?attacker ?]
             [attacker-download-server ?attacker ?download-server]
             (?malware-directory ?download-server.malware-directory)
             (?malicious-file ?malware-directory.files)
             (?startup-folder ?victim-computer.os.startup-directory)
             )
  :typing ((?victim-computer computer)
           (?victim-user user)
           (?victim-os windows)
           (?malicious-file file)
           )
  :plan (:sequential
         (:goal [achieve-remote-execution ?victim-computer ?victim-user])
         ;; action to add file into startup folder?
         (:goal [achieve-access-right write ?startup-folder ?victim-user])
         (:action [store-file ?victim-user ?startup-folder ?malicious-file])
         )
  :post-conditions ([has-persistent-remote-execution ?attacker ?victim-computer ?victim-user])
  :attack-identifier "T1547.001"
  )
```

Figure 4-8: Startup Folder Attack Method

Figure 4-8 shows the attack method for the startup folder attack method. Once again the malicious persistent file is defined in bindings. The attack method process has the initial goal of achieving remote execution. Next, the attacker will need to get write access to the folder by the goal achieve-access-right. Lastly, the attacker will have to store the file in the startup folder.

# Chapter 5

# Summary and Closing Remarks

## 5.1 Contributions

Expanding upon the knowledge base of the AttackPlanner, I was able to add some of the main prevalent persistence tactics according to MITRE. By adding this family of persistence methods along with the functionalities for these persistence methods to work, many more possible attack plans have opened up for the AttackPlanner.

## 5.2 Use Cases of Attack Planners

Attack plans can have many use cases. The first of which is that it helps with system auditing. Being able to run a system against an attack planner, whether it is individual attacks or a suite of attacks, it is immensely helpful in knowing the vulnerabilities within the system. Another use case is penetration testing. CALDERA, a tool developed by MITRE, was started to be used in conjunction with the AttackPlanner. Sam Dorchuck, a former Masters student at MIT, had a thesis detailing being able to parse an AttackPlanner's attack plan into an input that CALDERA would accept. From there, CALDERA would build an environment based on the parsed input and run automated tests of attacks against a given network topology. This is definitely a step in the right direction when it comes to automating penetration testing, as it would save a lot of time and money. The third use case is for hunting for attack plans. At the Defense Advanced Research Projects Agency (DARPA), there is a Cyber-Hunting at Scale (CHASE) program whose goal is to develop automated tools to find and detail attacks, gather data surrounding the attacks, and formulate defensive measures against these attacks. Creating defensive measures against attacks is also called mitigation planning, which is another possible use case. An attack plan is very useful when it comes to detailing attacks and aiding in defending against attacks. A fourth use case is that it can aid in system redesign. By knowing what vulnerabilities to look out for via attack plans and attack graphs, system designers can modify their existing systems to patch these vulnerabilities. Lastly, there is risk analysis. There have been research groups that have been using attack plans to analyze a system's risk factors, which is seeing how prone these systems are when it comes to system design. As a result, a conversation of trade offs ensues. There may be ideas to focus more on fault tolerance, but a side effect of that would be a lesser focus on simplicity. A lot of factors to consider when it comes to system design, but attack plans are definitely a good tool to aid in the decision making process and weighing risk factors.

## 5.3 Future Work

There are a few extensions of the AttackPlanner that are possible. The first of which is further systematization of the AttackPlanner. Expanding upon the knowledge base of the AttackPlanner is always needed for the improvement of the AttackPlanner. However, doing this manually can be a daunting task for the entire MITRE ATT&CK matrix. Thus, the question of automating this process has come up. Some of this involves scraping the MITRE website as well as doing natural language processing to analyze the wording of the description of attacks and convert them into the language of the AttackPlanner. A third extension is to fully automate the process of using the AttackPlanner in conjunction with CALDERA to penetration test. This involves automating the generation of attack plans, parsing and delivering them to CALDERA, and having CALDERA run goal-directed systems testing.

# Bibliography

[1] Michael Lyle Artz. *NetSPA: A Network Security Planning Architecture*. PhD dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 2002.

[2] Ulle Endriss. Search Techniques for Artificial Intelligence, 2005. PowerPoint slides of Prolog.

[3] Gregory J. Falco. *Cybersecurity for Urban Critical Infrastructure*. PhD Dissertation, Massachusetts Institute of Technology, Department of Urban Studies and Planning, June 2018.

[4] Malte Helmert. An Introduction to PDDL, October 2014. PowerPoint slides of Planning Domain Definition Language.

[5] Terrance R. Ingoldsby. Attack Tree-based Threat Risk Analysis. Technical report, Amenaza Technologies Limited, 2021.

[6] ISOGRAPH. Attack Tree Threat Analysis Software, 2019. Documentation of ISOGRAPH.

[7] Microsoft. Dynamic-link Library Search Order, July 2021. Online site for Windows DLL Search Order.

[8] The MITRECorporation. Persistence, October 2018. Online site for MITRE's Persistence Tactics.

[9] The MITRECorporation. CALDERA, March 2021. Documentation for MITRE's CALDERA.

[10] The MITRECorporation. MITRE ATT&CK Matrix, November 2021. Online site for MITRE ATT&CK Matrix.

[11] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. Mulval: A Logic-based Network Security Analyzer. *14th USENIX Security Symposium*, 2005.

[12] Steve Rowley, Howard E Shrobe, Robert Cassels, and Walter Hamscher. Joshua: Uniform Access to Heterogeneous Knowledge Structures, or Why Joshing is Better than Conniving or Planning. In *AAAI*, pages 48–52, 1987.

[13] Carlos Sarraute. Automated Attack Planning. *CoRR*, abs/1307.7808, 2013.

[14] Carlos Sarraute, Gerardo Richarte, and Jorge Lucángeli Obes. An Algorithm to Find Optimal Attack Paths in Nondeterministic Scenarios. *CoRR*, abs/1306.4040, 2013.

[15] Bruce Schneier. Attack Trees. *Dr.Dobb's Journal*, 1999.

[16] Oleg Sheyner, Joshua Haines, Somesh Jha, and Jeannette M. Wing Richard Lippman, editors. *Automated Generation and Analysis of Attack Graphs*, Security and Privacy. IEEE Symposium, May 2002.

[17] Kelly Shortridge. Deciduous: A Security Decision Tree Generator, July 2021. Documentation of Deciduous.

[18] Howard Shrobe. Computational Vulnerability Analysis for Information Survivability. *AI Magazine*, 23(4):81, 2002.

[19] Reid Simmons. Planning, Execution & Learning: Hierarchical Task Net Planning, September 2001. PowerPoint slides of Hierarchical Task Net Planning.

[20] Richard Skowyra, Steven R. Gomez, David Bigelow, James Landry, and Hamed Okhravi. Quasar: Quantitative Attack Space Analysis and Reasoning. Technical report, MIT Lincoln Laboratory, 1988.

[21] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Automated Generation of Attack Trees. Technical report, 2014 IEEE 27th Computer Security Foundations Symposium, July 2014.

[22] Jeannette M. Wing and Oleg Sheyner. Tools for Generating and Analyzing Attack Graphs. *FMCO*, 2003.