FIDIUS: Intelligent Support for Vulnerability Testing

Dominik Elsbroek and **Daniel Kohlsdorf** and **Dominik Menke** and **Lars Meyer**

{elsbroek,dkohl,dmke,lmeyer}@tzi.org
Center for Computing and Communication Technologies
University Bremen

Abstract

Security consultants are often hired to detect attack vectors of a client's computer network. The FIDIUS system is designed to support security consultants performing security checks of a computer network. By following a plan defined by files containing PDDL or intelligently choosing the next victim, the FIDIUS system is able to infiltrate a computer network automatically or control a specific host and check an IDSs' output for resulting incidents. We report first results of the FIDIUS project, the basic idea behind it and describe the current implementation of the system.

1 Introduction

Over the last decades computer security has become more and more important to companies and governments. Nowadays industrial espionage using security issues in computer networks is increasing. Also computer sabotage is a serious threat, not just since the emergence of Stuxnet (Schneier 2010). Companies hire professional penetration testers to elicit and fight these threats. These professionals examine the security condition of the companies' network, operating systems and used software. The penetration testers are security professionals with a corresponding knowledge about security and their tool collection.

Depending on the scenario the penetration tester will use different tools and methods to find possible attack vectors.

A professional intruder (also called "white hat") is for example consulted to check the offered services reachable through the Internet. The job of the white hat in this scenario is to audit the security and the possible attack vectors resulting from the networks' setup. Such services could be a mail service offered by a mail server, a web-service offered by a web server or a special service offered by a proprietary software.

To proceed with their security checks they will conduct several steps. They will collect basic information about the network first. Including addressing scheme, its differing zones or subnets and installed security measures. These could be intrusion detection systems or firewalls. Second he has to elicit knowledge about services used in the network and its hosts offering these services including their specific software version. Third, the consultant has to find out the vulnerabilities and attack vectors resulting of the network specific setup and the offered services within the network. To create a proof of concept for different attack scenarios the consultant could, if agreed, break into systems and demonstrate access to some documents as evidence for his success.

For every step mentioned above there are several public available tools to simplify these tasks. Common tools for host exploration within network systems from in- and outside are often used. Such a library will include active network scanners like *NMap*[1] and also passive tools like *p0f* ("Passive OS Fingerprinting")[2]. NMap is a tool for active network scanning standard coming with many build-in scanning types and a scripting engine.

After exploring a network with its services, vulnerabilities for specific software versions of offered services can be checked against contents of vulnerability databases such as the CVE (Common Vulnerabilities and Exposures)[3] and by searching public available mailing lists[4]. The latter ones also cover up-to-date information about known and newly discovered vulnerabilities (namely the Full Disclosure mailing list[5]). The found vulnerabilities can either be tested by using exploits coming with these vulnerability reports (or any other public available resource) or by the exploits and tools coming with a framework such as Metasploit (MSF)[6].

The MSF comes with a great number of exploits, payloads, scanners and fuzzers[7] which are also a common type of tool to check the robustness and thus the security of software (see (Dai, Murphy, and Kaiser 2010)). In section 5.3 we will discuss this framework in more detail.

These and many other tools and resources are part of the usual equipment of a security specialist (see (Bhattacharyya and Alisherov 2009)).

With the FIDIUS system we aim to support two scenarios. The first is a "gray box" scenario in which users have basic knowledge about assets and the net's topology. The second is a "black box" scenario in which the user wants to explore an unknown network without knowledge about the net. We propose two different agents, one for each use case.

[1] *http://nmap.org/*
[2] *http://www.cougarsecurity.com/p0f*
[3] e. g. US-Cert *http://www.us-cert.gov/* or *https://cve.mitre.org/*
[4] *http://seclists.org/*
[5] *http://seclists.org/fulldisclosure/*
[6] *http://www.metasploit.com/*
[7] *https://secure.wikimedia.org/wikipedia/en/wiki/Fuzz_testing*

Both are described later (see Section 5.2). For us, "gray box" users are system administrators using the system along with a net plan. For them the tool will find real attack vectors. The "black box" users are security experts, hired to break into a computer networks. In this case most properties of the net are unknown and the agent has limited knowledge.

2 Motivation

Black hats[8] attacking a network have unlimited time. In contrast, security consultants have a limited time budget because of financial constraints. Thus it is desirable to have software which is able to support. Such a software should support its user by automatically executing steps or suggesting next possible steps.

This software could not only been used by hired security specialists. Companies are also interested in testing how perceptive and sensitive their intrusion detection systems (IDS) are. Additionally scientific projects such as FIDeS[9] — which are aimed to create a new generation intrusion detection system — could be compared to common Security Incident and Event Management Systems (SIEMS) like ArcSight[10] and Prelude-IDS[11] in conjunction with Snort[12] and e. g. Prelude-LML or any other sensors which are able to report incidents using the IDMEF[13]. A software which can automatically intrude into a network could be used to compare different SIEMS.

FIDIUS[14] is such a software. FIDIUS is a framework which combines methods of the artificial intelligence (AI) with tools and frameworks designed for vulnerability testing. The architecture will be described in the sections 4 and 5.

Security specialists who have their own tools and processes are enabled to model them into the FIDIUS architecture. Or if these tools are even better located within the MSF they can use their tools nevertheless since FIDIUS uses a specially designed interface to the MSF using DRb[15]. Both ways should make their work more comfortable and efficient.

What is needed is a tool assisting a security consultant and a tool which automatically checks if an intrusion detection system recognizes all incidents. FIDIUS is designed to do both. With a component we call *EvasionDB*[16] we are able to examine which exploits will cause events thrown by an IDS. We count how many events are thrown with which severity level. This enables the AI to become more stealthy while blackbox testing. Furthermore FIDIUS is able to let security specialists model their procedures to intrude a network in the Planning Domain Definition Language (PDDL). The modules coming with FIDIUS can efficiently parse PDDL files, execute the defined steps and check the results. This makes it easy to create a set of repeatable tests which can be run like unit tests known from software development.

Additionally we have an AI component working with a neural network. It can be trained just by taking action with the graphical user interface. This AI component is tracking the steps taken by the user and can learn from these.

All the mentioned abilities of FIDIUS can help to save consultants time. It also enables security experts to share their knowledge by exchanging PDDL files. In addition FIDIUS can be used to test the configuration of a companies' IDS, IPS or DLPS.

3 Related Work

There are scripts and tools available which are able to attack a network nearly without user interaction. The free version of the Metasploit framework comes with a script called *db_autopwn*. This script attacks a host or a subnet with available exploits matching the operating system (OS) after scanning these hosts. This causes many events generated by an IDS though there are some options for fine tuning.

There is also the Metasploit framework in its professional version[17]. It enables the user among other features to create tasks which can be repeated automatically.

The novel approach of FIDIUS is to assimilate AI methods into such a system. Inspired by the intrusion detection systems extended by methods known from the research in AI we have played with some different methods like planning, neural networks and anomaly detection. Our goal is to automate attacks and integrate the planner into a testing tool. Therefor we needed a more flexible domain using sensing actions. The results of our approach are similar to the attack trees of (Junge 2010).

One of our AI components is based on "Action State Planning" using PDDL. An early approach for planning in vulnerability testing is described in the work of (Boddy et al. 2005). The domain includes social engineering and was designed for classic planners. The authors assume an omniscient attacker for their domain also known as whitebox testing.

Another approach was published on SecArt10 by (Jorge Lucangeli Obeset al. 2010). The authors presented a solution for transforming information from a pentesting tool into a planning domain scaling very well on medium sized networks. The precondition in this paper is also a whitebox scenario.

4 Architecture Overview

FIDIUS uses the MSF as a basis since it provides a vast list of exploits and payloads for different operating systems

[8] This term refers to a computer hacker and comes from the western movie's evidence being a "bad guy" when wearing a black hat. This term is used unlike "white hat" as an intruder in a good manner for hackers breaking into networks with bad intention. See also *https://secure.wikimedia.org/wikipedia/en/wiki/Black_hat*.

[9] Early Warning and Intrusion Detection System Based on Combined AI Methods, see *http://www.fides-security.org/*

[10] *https://www.arcsight.com/*

[11] *https://www.prelude-ids.com/*

[12] *http://www.snort.org/*

[13] see RFC 4765

[14] formerly *FIDIUS Intrusion Detection with Intelligent User Support*, see *http://fidius.me/*

[15] see *http://segment7.net/projects/ruby/drb/introduction.html*

[16] *http://fidius.me/en/news/released-rubygem-fidius-evasiondb*

[17] *https://www.rapid7.com/products/metasploit-pro.jsp*

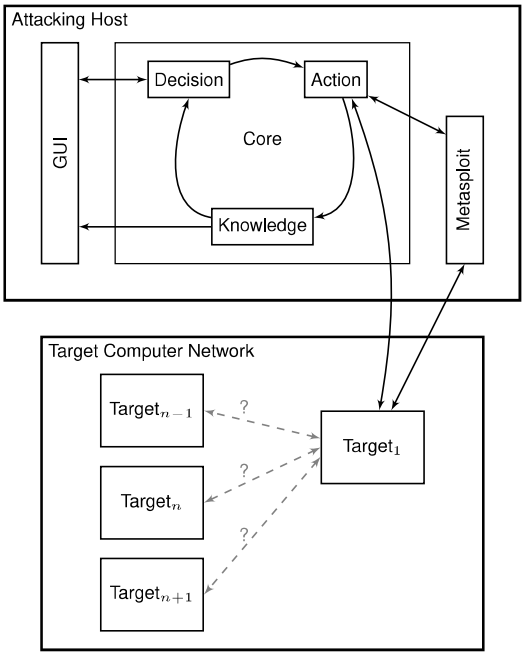

Figure 1: FIDIUS architecture overview

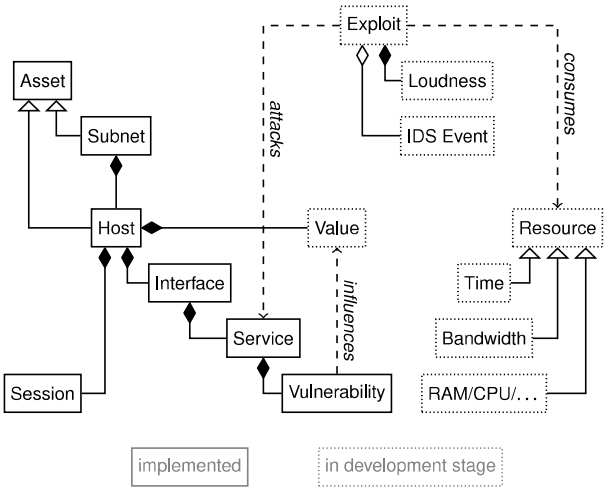

Figure 2: Current and future relationships in the FIDIUS Knowledge

and architectures. On top of this framework FIDIUS defines an architecture model as another layer which is inspired by the Model-View-Controller (MVC) design pattern. We use a *Knowledge Base* as Model, actions (i. e. scans and exploits with payloads) are taken by classes dedicated to the *Action* component. The *Decision* classes are able to use the AI methods and act as an interface to the GUI.

The actions taken by classes placed in the *Action* part result in changes within the knowledge base: Newly discovered hosts will affect the knowledge base since there might also be new services with specific versions etc. The decision component observes the knowledge base and might alter the planning actions (if the newly acquired knowledge indicates this to be useful).

This structure makes the architecture of FIDIUS very expandable and flexible. Third parties are invited to implement their own action or decision components.

4.1 Knowledge

As introduced above, the *FIDIUS Knowledge* is figuratively equivalent to the "model" part in the MVC design pattern.

So the knowledge component (cf. Figure 2) contains various information about *Assets*. An Asset may be a single *Host* or an entire *Subnet* and may provide many different *Services* on different *Interfaces*. A Service in return could have known *Vulnerabilities*. A Host may has also zero or more *Sessions*, which indicates a previous (and/or active) intrusion. Any host has a specific *Value*, which is estimated by the running services and hidden information, such as the number of interfaces or the amount of free disk space.

Besides these asset-related information, there is an exploit-related view, which is currently in discussion. Targeting a barely observable intrusion in a computer network, there are two factors making this actions observable: On the one hand there is something like "*Loudness*" as a result of incidents thrown by an IDS and on the other hand there are "*Resources*" consumed. The main concepts behind this discussion will be further described in section 4.3.

4.2 Decision

The *Decision* component is responsible for planning the next steps based on the current knowledge. Decisions can be made either by an intelligent agent or a user by using the user interface (see section 5.1). A decision component can read from the knowledge database (see section 4.1) and trigger *Actions* to be performed (see section 4.3). Furthermore, changes in the knowledge database are recognized and the agents are updated accordingly.

4.3 Action

A module or class placed in the *Action* component is built to take actions on the network to be attacked or a host located within the network. It corresponds to the "control" part in the MVC pattern. Actions are taken on networks or hosts to gain knowledge or (almost unauthorized) access to an asset.

Since all actions consume resources it is useful to have a measurement making actions comparable w. r. t. their usage of bandwidth, CPU, RAM etc.

The MSF uses "jobs" as an indicator for resource consumption of tasks without any differentiation. But running a web server with a website containing malicious code is not as resource consumptive as running a network scan with NMap. Also waiting for someone falling into a trap is not as visible for some IDSs as sending an exploit downloading a payload having a size of many megabytes after successfully exploiting a vulnerability. Thus we have defined some more meaningful attributes. They will be described in the following section.

Resource Consumption To evaluate the costs of an action we have to consider the consumption of CPU and RAM on both sides since an increasing RAM usage could alert Host Intrusion Detection Systems (HIDS) or monitoring tools like Nagios[18] or Munin[19]. One should also consider the count and size of packets need to be sent to get an exploit working. The latest exploit for the Exim4 mail transfer agent for example, needs an email of at minimum 50 megabytes to be sent[20].

To distinguish the different types of resources we have created classes which are listed below.

Bandwidth usage: The amount of traffic sent over a network by an action should be known. Protocols like Netflow or IPFIX (IP Flow Information Export) are able to account traffic to hosts and sensors watching parameters like "traffic per hour" could generate alerts or even warnings. Anomaly based IDS also could generate a warning or even an alert if a host produces abnormal amount of traffic.

Host resources: The execution of actions will consume resources of both the local and remote host, such as time and bandwidth available by NICs (Network Interface Card) but as mentioned above also CPU, RAM and also additional use of memory on hard disc drives is possible. Also the count of started processes can be monitored by corresponding tools and thus should be taken into consideration.

Intrusion detection: An exploit sent over the network detected by an IDS is *very loud*[21]. This factor is also hard to determine, since almost all rule based IDS have different rule sets and anomaly based IDS have different definitions of normal traffic. Additionally both kinds of IDS can have different configurations[22].

Value/cost ratio: The decision to attack a specific host will be made based on measurable factors. The precedence of the obvious factor — the host's value — should be lowered down, if an exploit has a high resource usage and/or has a great loudness.

[18] *http://www.nagios.org/*

[19] *http://munin-monitoring.org/*

[20] see CVE-2010-4344, *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4344*

[21] in sense of many generated alerting events

[22] In FIDIUS a group started with "evasion testing" to find out which exploits (and their configuration) will not be detected by intrusion detection systems. Therefore, exploits are mapped to IDS alerts and saved as knowledge in a database.

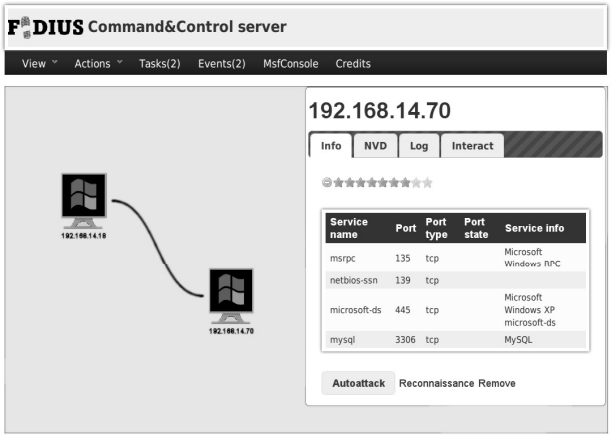

Figure 3: Screenshot of the FIDIUS Command&Control server

Not all of these factors are measurable exactly. For example how much RAM or CPU the execution of an exploit will use on the remote host will depend on different factors. Also the generated network traffic and its conspicuousness will depend on the network and its daily usage. Measuring these factors and taking them into account for decisions taken by the AI have to be done. They are mentioned here for the sake of completeness since IDSs can also take these values as indicators for intruders.

5 Current Implementation

5.1 User Interface

The FIDIUS user interface (see Figure 3), called *Command&Control server*, is basically a (quite simple) web application, which displays the knowledge (see Section 4.1) in form of a network graph.

The Command&Control server itself is written in Ruby using the Ruby on Rails framework[23] and is connected to the architecture's core through a XML RPC interface. The latter fact means two important things:

1. The GUI does not need any database to manage running attacks, information on overtaken hosts, etc.
2. The Rails application might be replaced by any other preferred GUI toolkit.

The AI component (and action component, implicitly) won't run any step without user confirmation for security reasons.

5.2 Artificial Intelligence

We implemented two intelligent agents for our decision component. One using "action state planning" for attack plan generation, the other for the prediction of a host's value. Depending on the scenario and the user we recommend to choose one of these agents. For the "gray box" scenario we implemented an "action state planning" agent and for the

[23] *http://www.rubyonrails.org/*

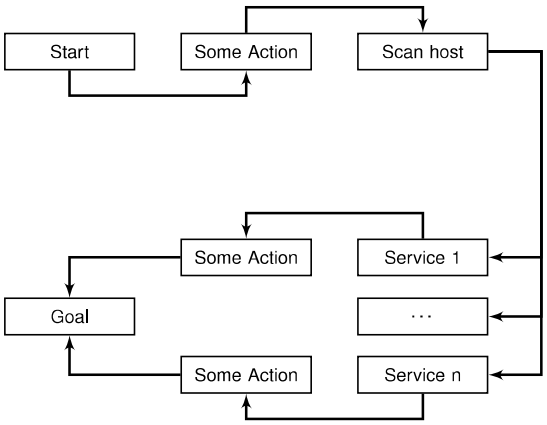

Figure 4: A contingent plan branching after a sensing action

"black box" scenario a "host's value prediction agent" which is a predictive artificial neural net adopting to the users preferences using NMap scan data. In the following we will describe how the agent works.

Action State Planning Agent In the beginning the agent reads all the relevant information from the knowledge database and converts it into PDDL using the following main predicates:

1. **onHost**(X) Our agent is currently on host X.
2. **hostVisible**(X, Y) A host Y is visible from host X.
3. **serviceRunning**(X,Y) Is a service Y running on host X.
4. **inSubnet**(X, Y) Host X is located in subnet Y.

The domain also supports knowledge about running IDSs and Anti Virus software. The planner avoids subnets with an IDS running and hosts with Anti Virus software. Furthermore, the domain contains actions and predicates for plumbing[24], password cracking and domain controllers. If a computer is plumbed, the password cracked or it is in the same subnet with a domain controller, it can be accessed without exploiting it.

All objects in the domain are typed. Existing types are computers, services and nets. The computer type is split into clients and servers.

In our implementation we have two existing methods for plan generation. One for normal plans, the other for contingent plans. For the classical planning we use Hoffmann's "Fast Forward" (FF) (Hoffmann and Nebel 2001) planner and for the contingent plans the "Contingent Fast Forward" (cFF) (Hoffmann 2005) planner. When using the classical planner the user has to specify all hosts, its connections, its subnets and its services in advance. When using contingent planning the user needs the same specification as above except for the services per host. In this case the user specifies in which services he is interested in (he knows an exploit for) and the agent includes a sensing action for these services called "scan". The resulting plan is different when including such actions. The result of the FF planner is a list of actions leading to the goal. cFF results are also a list of actions, except for the sensing actions. In a sensing action the planner creates a branch for each possible outcome of the action (see Figure 4). In our case one branch per possible service. Another sensing action is checking for a web server on a host. The benefit of sensing actions is that we do not have to know everything in advance. For example services might change very quickly while the main net's topology remains. So excluding knowlede that changes often allows more dynamic plans. Furthermore, listing all possibilities in a contingent plan has the advantage that we can decide how to scan or avoid scanning. The information about running services can be derived from context information. For example established connections from one host to another are sufficient indicatiors for the services connected to. Another example is passive scanning by sniffing pakets from the network or monitoring connections using netflow.

The types used in the domain are computers, services and nets. Furthermore there are servers and clients which are of the type computer. The current location of the attacker is given by the host he is currently operating on and the subnet of this host.

The important predicates of our domain for common goals are:

1. **server_owned**(X): Have we exploited a server in net X?
2. **iframe_injected**(X): Do we have an iFrame trap in a net X?
3. **plum_in**(X): Do we have the opportunity to come back in a subnet X through a back door?
4. **powned_dc**(X): Do we have control over a domain controller in a net X?

These predicates are mainly used to create our goals. In some cases the goal might be to hack a specific host, so the goal is specified using the on_host predicate. Another example is to conquer an important source of information in a specific subnet. Assuming that such sources are normally servers, we can use the $server\_owned$ predicate. If we want to install a trap for users of a subnet the goal might be to inject an iframe in this net ($ifrmae\_injected$ predicate). The last example is the option for a return in an exploited net. In this case our goal is to conquer a domain controller or install a back door in the net.

The plan can use the following main actions to accomplish a goal:

1. **Move** Moving from host X to host Y. If Y is visible from X, Y is exploited and the agent has control over host X, the agent can perform this action. The result is the change of location from X to Y.
2. **Scan** The sensing action. Scanning a host Y being on host X. This action is possible if the agent's location is host X and host Y is visible from X. It observes if a service S is running on host Y.
3. **Exploit** Exploiting a host Y being on host X using an exploit for service S. Therefore the agent has to be on host X, host Y is visible from host X and the service S is running on host Y.

[24] We exploited the host before and installed a backdoor.

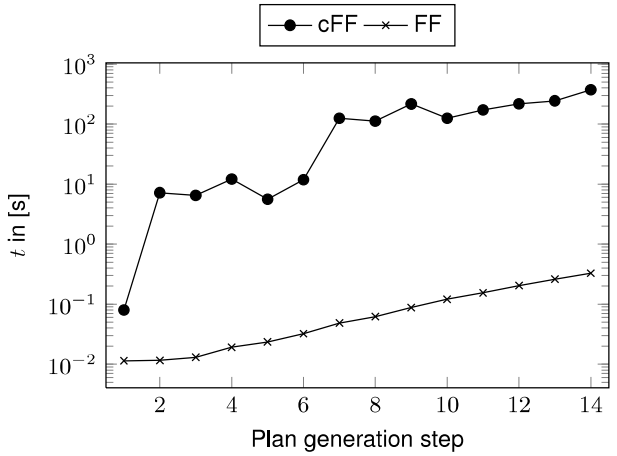

Figure 5: The result of the performance experiment for the FF and the cFF planner

Furthermore, our domain includes actions for different exploiting actions like password cracking, installing back doors or iframe injection. When a password is known for a host or a back door is installed on it, the planner mustn't exploit a host first. Furthermore we have the possibility to include knowledge about domain controllers, NIDS and Anti Virus Software. When a domain controller is hacked, all hosts in the same subnet can be accessed without exploiting them first. Subnets including an NIDS and hosts that include Anti Virus Software, are avoided as targets by the planner.

As indicated above the agent plans all the steps beforehand. During execution the agent takes the next action in the plan, executes it using the associated functionality from the *Action* component and evaluates the results. Because the agent plans everything in advance (except for services in the cFF version) it can't be used in a black box scenario. Because the agent will be integrated in an interactive tool we did a performance experiment. In the experiment we generate random computer networks with n nodes, each with a branching factor of b and m services. Then we executed the planning and measured how long it took the agent to generate a plan. This procedure was repeated while increasing n, m, b by doubling.

As one can see in Figure 5 the cFF planner is slow. The value added by cFF is the possibility for sensing actions. In our case this are nmap scans. If one is willing to provide all running service per host to the planer in advance the agent can use the faster FF planner.

Host's Value Prediction Agent In a black box scenario our planning approach is not an option because all the knowledge has to be known in advance. So we developed an agent which decides the next host to exploit online using an artificial Neural Net. When a new host is added to the knowledge base, it is scanned using the NMap action component. The agent inserts all ports in a feature vector $X = (x_1, ..., x_n)$ with $x_i \in \{0, 1\}$ for $i = \{1 \dots n\}, n \in \mathbb{N}$. The index i refers to a port number while $x_i = 0$ indicates an open port and $x_i = 1$ a closed one. Furthermore, a user specified value v is assigned with each host. We train the neural net with a training data set, where each instance is a tuple of (X, v) in order to predict v for an input X. The training is performed in $100-400$ iterations. The net's topology is an input layer of the size of the feature vector, $m \in \mathbb{N}$ hidden layers, where m is a user specified parameter with default value $m = 10$. The output layer is just one node. We implemented the net using the `ai4r` library[25].

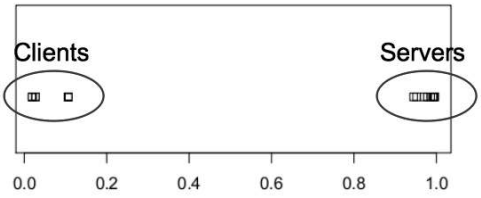

Figure 6: Where the Neural Net maps clients and servers.

During a session the net predicts the value for each host and inserts it into a priority queue. The host with the highest value-prediction is the next host to exploit. Therefore, each "host value" tuple is inserted into a priority queue during run time. The priority queue used is from the ruby `algorithms` library[26]. If the user disagrees with the prediction he can adjust the predicted value and the net is retrained.

In an experiment we collected a small data set of hosts' service vectors. Our use case was a user preferring to exploit servers. So all client's values in the data set were set to 0 and all server's values to 1. The prediction error (mean square error) is 0.03 so the prediction is working. In Figure 6 we can see that the Neural Net is mapping clients near to 0 and servers near to 1.

In a second scenario the intruder wants to gather user data from hacked hosts. We assume that user data can be found in data bases and on the clients. So we mapped all values of clients and all servers with a data base to 1. We mapped all other host values to 0. The prediction error is 0.035, so our method is also working properly in this case.

In the last scenario the intruder wants to obtain access to mail servers. So we set all servers with an open mail port to 1 all other hosts to 0. The prediction error is higher then in the previous scenarios but is also small 0.15.

Our results indicate that the neural net can predict user's preferences if the preferences follow a specific pattern. In our experiment the classes are linear separable. However, Neural Nets are able to adopt to complex non linear functions. So we think this method will also work for other user preferences following a specific pattern.

[25] *http://ai4r.rubyforge.org/*

[26] *http://www.igvita.com/2009/03/26/ruby-algorithms-sorting-trie-heaps/*

5.3 Metasploit framework

The Metasploit framework (MSF) has originally been developed as a framework which enables exploit developers to concentrate on their specific work by providing often required functions, protocols and configurable payloads which are in nearly every exploit the same. In fact, most people want an exploit which automatically opens a port with a command line interface on the remote host after successfully exploiting a host. This is given as a payload by the MSF whereby the port is configurable as an option of the payload.

Nowadays, the MSF is basically a large penetration testing framework which is open-source and contains several exploits for various kind of software. In general, most of the security issues which are used by the exploits coming with the MSF are fixed on an up-to-date system, but there is a good chance that the recent ones will work very well. In fact, most exploits published on security mailing lists[27] will be integrated into the framework short time after certain information is available. For our purpose, testing different IDSs and their configuration, there are also websites hosting old versions of different services. So its also possible to run services with outdated software versions to check what kind of attacks and which steps of an attack are recognized by an IDS.

Besides these exploits, the MSF contains a lot of helper classes (called "auxiliaries"), for many purposes like application version detection or password sniffing. Other useful auxiliaries are different vulnerability scanners, brute-force password cracking tools or a set of simple server systems. In addition, the MSF provides output obfuscation — which allows outsmarting most of the string-matching based detection systems.

Attacking a host means basically to gain access to the operating system's command line interface. But the command line interface differs by operating system. Therefore, the MSF provides a special payload called "Meterpreter"(Miller 2004). It abstracts the operating system's command line interface and brings different advantages. First, it allows to run arbitrary commands without starting a new process, so that *host-based intrusion detection systems* (HIDS) do not recognize an increasing process count. Second, it provides an operating system independent high-level shell for the attacker. This will allow the Meterpreter either to gain higher user privileges or to exploit other hosts. Third, the Meterpreter may also run "Meterpreter scripts" to automate some steps again without creating any recognizable processes.

Usage in FIDIUS The MSF requires manual interaction for each step (i. e. run auxiliaries, prepare and run an exploit, analyse the exploit's output, etc.). Also the framework itself doesn't have a detailed knowledge of the network setup it attacks. Fortunately, the MSF has an API to use the framework not only as an application, but also as a library. There are still some disadvantages, e. g. the MSF requires all modules (including exploit, payload etc.) to have a specific format. The framework then loads all these module files into a manageable format. All modules are loaded into main memory on start up. This results in an almost unacceptable loading time especially while developing new features. Testing any tool using the framework as library can be very time exhausting this way. In FIDIUS we separate the MSF in another application so that the framework is available through network connections.

Thus in the *Action* part of *FIDIUS Core* there is an exploiting part and a scanning part. The exploiting part is an extended and cleaned version of the *db_autopwn* method of the MSF. We have therefore sorted the exploits by date as a first approach. So the order of exploits run against a service starts with the most recent ones. First this results in a much better performance. With a time ordered list of exploit it took us five tries to exploit a "Windows XP with SP2". With a list of exploits only ordered by operating system the amount of exploits tested is 26. Second this results in a much lower resource consumption such as network bandwidth and CPU load. And third, sending only five exploits through a network which is possibly monitored by an IDS will result in much less events thrown by that IDS than sending possibly 40 or even more exploits including shell-code and suspicious OS-API calls.

For the actions to scan hosts or networks we have written different classes implementing different kinds of scans such as ARP scan, Ping scan or port scan. We also changed the OS version detection part thus we can use exploits matching the given hosts set up in more detail. Thus our actions become much more silent measured by the parameters described in section 4.3.

To decrease the great loading time we use the `msfdrbd` *(Metasploit Distributed Ruby Daemon)*. It uses the Ruby standard library `DRb` *(Distributed Ruby)*[28] which allows the bidirectional sharing of (binary) objects (even on different machines). In this way, the `msfdrbd` can constantly run in a background process, instantiate the Metasploit framework and wait for incoming requests. Thus we don't have to wait for all modules to be loaded by the framework while developing the *FIDIUS Core*. Furthermore we have a more scalable software design. Additionally a simple control interface allows the user to interact with the daemon to load and unload server-side framework plug ins. Thus, the execution of exploits will be simplified by providing wrapper methods.

6 Conclusion & Future Work

We presented an architecture for an intelligent vulnerability testing tool and its current implementation. In the architecture part we described the Knowledge, Decision and Action components. The current implementation using *FIDIUS Core* is a user interface, a planning component, a neural net and an exploiting component using the MSF.

In the future, the final FIDIUS system should be an improvement for both penetration testers and network administrators. It simplifies their work and saves their time.

There are still many things we would like to implement and improve. The recognition and definition of the value

[27] not only Full Disclosure, *http://seclists.org/fulldisclosure/*, but also the Metasploit mailing list, *http://mail.metasploit.com/mailman/listinfo/framework* and others

[28] see *http://segment7.net/projects/ruby/drb/introduction.html*

of a host and an exploits loudness have to be fine-tuned. This is important to improve the results of the used AI algorithms. Furthermore we need a bigger knowledge base with IDMEF events from different IDSs with different configurations. This would make an assigned loudness more reliable. We have started this by creating the *EvasionDB* mentioned in section 2 but there are still several tests to be done. In the future we plan to create a database connecting each exploit with all tested IDS wrt their configuration and the events caused by this exploit by an IDS with specific configuration. We plan do use this database to train a neural net. Such a database could probably also be used for other AI methods. This is most important for black box testing.

Additionally, we are still testing the adaptiveness of the neural net in a user study and will test the planner in preconfigured test nets.

7 Acknowledgement

FIDIUS is a project at the Department for Mathematics and Computer Science of the University Bremen. We want to thank all FIDIUS project members who worked with us and made this publication possible. Besides the authors this are: Christian Arndt, Andreas Bender, Jens Frber, Willi Fast, Kai-Uwe Hanke, Dimitri Hellmann, Bernd Katzmarski, Hauke Mehrtens, Marten Wirsik. Furthermore, we would like to thank the project's supervisors which are Carsten Bormann, Stefan Edelkamp, Mirko Hostmann, Henk Birkholz and Jonas Heer.

We also want to mention and to thank the FIDeS project for various inputs and providing a network for testing purposes.

References

Bhattacharyya, D., and Alisherov, F. A. 2009. Penetration testing for hire. In *International Journal of Advanced Science and Technology*, volume 8.

Bishop, C. M. 2007. *Pattern Recognition and Machine Learning*. Springer.

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In *ICAPS 2005*, 12–21. AAAI Press.

Lucangeli Obes, Jorge; Sarraute, Carlos and Richarte, Gerardo 2010. Attack Planning in the Real World. AAAI Press.

Dai, H.; Murphy, C.; and Kaiser, G. 2010. Conguration fuzzing for software vulnerability detection. In *ARES'10: International Conference on Availability, Reliability and Security*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. In *Journal of Artificial Intelligence Research*, volume 14.

Hoffmann, J. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS'05*, 71–80. AAAI.

Junge, F. 2010. *Dynamische Erstellung von Angriffsbäumen für Rechnernetze mittels eines modularen Werkzeugs*. Diploma thesis, Universitt Bremen.

Miller, M. 2004. Metasploit's meterpreter. *Metasploit documentation*. Online available at *http://www.metasploit.com/documents/meterpreter.pdf*. Accessed 2011/03/21.

Russel, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition.

Schneier, B. 2010. Stuxnet. *Schneier on Security (Weblog)*. Online available at *http://www.schneier.com/blog/archives/2010/10/stuxnet.html*. Accessed 2011/03/29.