

A Potency Relation for Worms and Next-Generation Attack Tools

Dan Ellis

12 March 2002

MITRE Technical Report

Submitted to RAID 2002

Abstract: We describe the attributes that span the space of attack tools, illustrate a class of next-generation attack tools and contrast it with contemporary worms, and outline a methodology that can be used to express the potency of the threat. In identifying what influences the potency of a threat we hope to identify underlying principles that govern the potency of a class of attack tools or a specific instance. This methodology is based on an analytical process that maps two spaces and illustrates the relationships between these spaces. The first space is described by the fundamental attributes that describe the space of attack tools. The second space describes various metrics of the potency of attack tools. The potency relation is a formulation of the relationship between attributes possessed by an attack tool and a description of how potent that attack tool is. This potency relation, therefore, can be used for both predictive as well as defensive purposes. By identifying what factors influence the potency of an attack tool, we have identified the very factors a defender must influence in order to minimize potency.

Key words: Worms, Viruses, Malware, Intrusion Detection, Modeling, Potency Relation

1.0 Introduction

The security community, in general, lacks understanding of the threats that worms and other mobile attack tools pose. Initial work on worm technology was performed over two decades ago [1] by looking at benevolent uses for worm technology. A definition of worms [2] that builds off a previous definition for viruses [3] was presented by Cohen, both being based on a Turing Machine model. It is not clear from either definition how to extract information about the potency of these classes of attack tools. Further, the definitions are sufficiently general to classify a wide array of threats into two large classes. The security community needs some way to further refine these two large classes of threats into identifiable threats and understand those factors that affect their potency.

Recently, two position papers [4, 5] have identified the need to describe the potency of a hypothetical, yet realistic threat. Each researcher proposed a variation on contemporary worms that, they claim, can significantly affect potency. The position papers underscore the point that the community lacks a clear methodology and metrics for reasoning about the potency of even contemporary worms. Cohen proposed an attribute that can significantly enhance the potency of an attack: coordination [6]. More work is needed to identify those fundamental attributes that govern the potency of attack tools.

The virus community has contributed some understanding of the potency of mobile attack tools at epidemiological levels. These models attempt to describe the behavior of attack tools that spread over large populations in ways that are easily and accurately captured in statistics. The models, however, have been criticized for not answering some fundamental questions [7]. Some recently proposed models answer some of the outstanding questions [8]. However, even these models do not adequately address attack tools whose behavior is not generated by probability.

The anecdotal evidence [9] of the last year shows that our defenses are not yet sufficient to counter even the threat of contemporary worms. Whereas our defenses will never be more effective than our understanding is complete, we need a way to reason about contemporary worms and other, more sophisticated, attack tools. Further, because worms are evolving rapidly, we need to be more forward looking and identify future threats to identify possible defenses before a crisis.

As a research community, we need to explore and attempt to answer several questions:

- How do you compare or measure the potency of a worm?
- How potent are contemporary worms?
- How are worms likely to evolve in the next few years?
- How much more potent are next-generation worms?

Until we have answered the preceding questions, we will not be able to effectively answer the following questions:

- What defenses are effective at detecting and protecting against contemporary worms?
- What defenses are necessary to combat next-generation worms?

From a more practical view, the ability to classify threats allows for more accurate allocation of resources to defend against in-bound threats. We pose the question:

- Given a large number of attacks and attackers, how do you allocate your resources to provide the most effective defense?

Current intrusion detection systems (IDSs) are overwhelmed by alerts. The intrusion detection analysts (IDAs) who operate the IDSs are inundated with information. Given a large set of alarms, identifying which alarms are of highest priority it is not trivial. Although current IDSs have some capabilities to prioritize alerts at the lowest level, they do not yet provide mechanisms that include a classification of the threat. Two different threats (e.g., worms) might both attack the same system vulnerability with the same exploit. However, one threat may be significantly more potent than the other. Determining potency and then responding accordingly, that is, with more resources allocated towards deterring the most potent threat, is critical to providing efficient and effective defenses. Current defenses identify only the respective potency of exploits, not the potency of the actual threat, the attackers.

While we necessarily devote significant resources to improving our understanding of systems, system vulnerabilities, and detection mechanisms, we also need to devote resources to improving our understanding of the threats. This author disagrees with the opinions of other researchers (as voiced in [8]) who believe that discussing threats such as worm technology will inflict more damage on society than silence. Though in attempting to understand the threat we may unwittingly assist malefactors enhance their trade, at the very least we are on equal grounds of understanding and may even have the advantage. Such is not the case today. Further, failure to understand the threats will lead to good security solutions that solve the wrong problem now and in the future.

In Section 2, this paper presents an enumeration of fundamental attributes relevant to the threat classes of attack tools and an associated classification scheme. The fundamental attributes are described, one of which, the rules-of-engagement policy (ROEP), is novel in the discussion on threats. After the groundwork has been laid, relevant background work is presented. Section 3 discusses potency metrics for classes of attack tools. Section 4 presents the analytical methodology embodied in the potency relation. Section 5 describes the class of smart worms and explains its significance. Section 6 briefly describes work in progress that attempts to answer these open questions.

2.0 Attributes of Attack Tools

In this paper, an attribute is a general description of a capability. Although the attributes selected to populate the attack tools attribute set can be arbitrary, it is important to identify those attributes which are most descriptive and of greatest interest. Similarly, it is important to choose attributes at the appropriate level of abstraction to provide the most simple, yet complete, space of classes of attack tools.

A set of attributes describes a space of classes of attack tools. The attributes of interest in this paper are mobility (MOB), modularity (MOD), coordination (CO), goal-directed (GO), and policy-constrained (ROEP). The set of attributes can be represented as {MOB, MOD, COORD, ROEP, GOAL}. Each subset of attributes describes a separate class of attack tools within the space of attack tools. For example, the subset {MOB} defines the class of attack tools that are mobile (i.e., most contemporary worms and viruses). The subset {CO} defines the class of attack tools that possess the ability to coordinate actions (e.g., many contemporary distributed denial of service [DDoS] attack tools). The set {MOB, MOD, GO} defines the class of attack tools that possess the capabilities of mobility, modularity, and goal-directed. Thus, the power set enumerates the classes within the space of attack tools. The set containing all of the attributes defines a particularly interesting class of attack tools, smart worms, described in Section 5.

2.1 Mobility

Mobility is the capability to instantiate a (possibly evolved) copy of oneself on a remote host and execute it. In this discussion, we are most interested in subjects that are active processes (i.e., worms), not passive files (i.e., viruses). Although both worms and viruses have the general capability of mobility, the latter is dependent upon an external force to trigger it, that is, to enable the virus to become active. The line between activity and passivity does become somewhat blurred as triggering actions for viruses become more automated or have higher frequencies of activation (e.g., Nimda). Another difference between worms and viruses is the domain of infected objects: worms infect hosts and viruses infect files. The difference in domain is easily abstracted away. Therefore, viruses, fundamentally, are special cases of worms that have periods of inactivity. Although there are implementation differences between viruses and worms, the general algorithmic phases of each are essentially equivalent.

Mobility is a fundamental attribute that is necessary to maneuver through a system. Not all vantage points are the same and not all attacks can be effectively executed by a single attacker. Mobility is the key to improving one's vantage point. Further, as mobility is the key to

multiplying the number of potential attackers, it is also the key to increasing one's force projection.

2.2 Modularity

Modularity describes many attributes; most are purely performance-related. Others, however, are fundamental. The fundamental aspects of modularity involve the ability to execute arbitrary commands dynamically and interface with arbitrary payloads. Performance-related attributes are not discussed here.

If an attack tool does not possess a mechanism for acquiring and executing arbitrary actions, then the attack tool relies entirely on a priori knowledge about what it will need to do. Conversely, an attack tool that is modular can learn about an environment and then request or devise a new activity to fit the need. This dynamic command execution allows attacks to proceed in ways not previously envisioned and releases the constraint of perfect a priori knowledge about the environment and objectives.

The ability to interface arbitrary payloads further enhances an attack tool's capabilities. The ability to interface any external program and use it as a tool allows the agent (i.e., the local instance of the attack tool) to execute any attack that can be executed in a program. Rather than having to imbed every conceivable attack into the agent, simply creating an interface between the agent and a tool set immediately enables the agent to take advantage of any tool in the tool set. Thus, from this attribute comes the generality necessary to execute any attack that is programmable.

This modularity attribute is the key to the generality of attack tools. As soon as an attack has been automated by some program and an interface has been created to leverage the tool from the agent's perspective, that attack can be executed by the agent. What remains, then, is a way to describe to the agent the appropriate time and manner to use the tool.

Some instances of modularity have been exemplified in historical attack tools. Tribal Flood Network 2000 (TFN2K) [10] is a distributed denial of service (DDoS) attack tool that provides a shell for executing arbitrary commands on the remote host. nessus [11] is a network-scanning and reconnaissance tool allowing users to create plugins that can be dynamically executed. Each plugin automates one scan.

2.3 Coordination

Coordination is the capability to organize and execute activities among disparate and independent agents in a meaningful way. It allows independent agents (subjects) to collectively decide on a set of activities or actions to be performed to modify the environment. An activity is a four-tuple $\langle \text{subject}, \text{object}, \text{action}, \text{time} \rangle$, where the *subject* is the initiator of *action* on some *object* at a specific *time*. The *action* may be a complex, parameterized activity.

Coordination can be performed either statically or dynamically. Static coordination implies that an agent's activities are determined during the attack initialization phase. Dynamic coordination, however, implies that an agent's activities can be determined at any point in time. Trivial, static

coordination has been performed by some historical worms (e.g., the Code Red worm). Slightly more sophisticated, dynamic coordination has been evidenced in contemporary DDoS attack tools.

One example of static coordination is the Code Red worm. The Code Red source code statically specified that, starting on the 21st day of the month, all agents begin a denial of service (DoS) attack on `www.whitehouse.gov`. The first twenty days of the month are to be spent by each agent propagating with no coordination. Thus, the coordination can be described with two activities: `< * , * , propagate , day_of_month[1 .. 20] >` and `< * , www.whitehouse.gov , DoS_attack , day_of_month[21 .. 31] >`, where `*` is the wild card. This coarse-grained coordination is the first observed in any historical worm.

Tribal Flood Network 2000 (TFN2K) is an example of an attack tool that uses dynamic coordination. A TFN2K zombie is an immobile agent that sits on a host and waits for an order to perform an activity. The agent knows how to perform a few types of DoS attacks, as they are statically defined in its source code. The agent waits for an order that clarifies which statically defined attack to execute, which host to target, and when (via a start or stop command). A typical order is to start (or stop) attacking `target.host.com` with a ping of death attack and is directed to the set of intended attackers. With a little effort (by using the blind shell command to self schedule a cron task, but not implemented by default in TFN2K), this can be extended to `<* , target.host.com, ping_of_death, 10:30GMT>`, where `*` is the wild card for the source(s) of the attack, `target.host.com` is the target of the attack, `ping_of_death` is the type of attack, and the last field indicates the timing of the attack. This type of coordination is more sophisticated in that there is some variability in the action parameter and arbitrary variability in the time parameter.

Both of the above examples illustrate simple coordination. Each action is not just context-free but stateless. More sophisticated coordination requires the ability to link activities into a chain and retain state information between activities. A scenario is a specified sequence of activities linked in a way that collectively resolves the parameters of the activities. Thus, a scenario is analogous to a state machine and each activity is analogous to a state transition.

An example of more sophisticated coordination is demonstrated with the following attack, actually executed against one of MITRE's computers. From the attacker's perspective, the attack is as follows:

1. Identify the host, A, running Remote Procedure Call (RPC) services.
2. Query A's portmapper to find out which host, B, is running the `snmpxdmtd` service.
3. Attack B's `snmpxdmtd` with a buffer overflow that has a backdoor in it.
4. Connect to B using the backdoor installed in step 3.

This attack requires the state to be maintained across actions. A single attacker can execute each of these actions from a single host, retaining the state between events. However, this attack can be distributed over four different attacking hosts. As an example, hosts W, X, Y, and Z could each perform steps 1, 2, 3, and 4, respectively. W would tell X which host (and port) to query as the portmapper for an `snmpxdmtd`. X would tell Y which host and port to attack with the backdoor. Y would tell Z when to connect to B using the backdoor. The amount of information communicated between the attacking hosts can vary, depending on the flexibility of the scenario. It is clear, however, that dynamic coordination of some sort is required for this to be performed

in a distributed manner. Whereas distributed computation is nontrivial, it is also clear that it would be much easier for the attacker to attack from a single host.

The question might be raised regarding the reason for doing something in a distributed manner when it can be performed without distribution. One significant reason is to make detection and attribution much more difficult. Avoiding detection concerns many attackers. Another reason for employing distribution and coordination is the use of an obvious force multiplier as the number of attacking hosts increases while synchronization is needed (such as for DDoS attacks). The attack performed on MITRE actually was performed using four different hosts.

2.4 Policy-Constrained

The capability of limiting an agent's behavior to a subset of its potential behavior can be accomplished using an ROEP. The ROEP dictates not what *can* be done but what *may* be done. The ROEP defines bounds to the activities an agent can perform. Before any activity can be performed, it must be either explicitly approved or rejected by the ROEP.

The advantages of using an ROEP are significant. Many attackers would like to be able to achieve an objective while satisfying certain other constraints. Many constraints might be of concern to an attacker. The ROEP is the mechanism that allows an attacker to satisfy those constraints. The following is a list of possible constraints. The attacker might want to perform an activity only if:

- The activity has a probability of being detected that falls below some threshold.
- The activity has a probability of being attributed that falls below some threshold.
- The activity has a probability of being successful that falls above some threshold.
- The activity produces no internet traffic that is visible to a machine in a particular domain (e.g., country).
- The activity produces no side effects of a particular nature.
- The order producing the activity has been properly verified.
- The activity satisfies legal or ethical constraints.

An attacker might desire that any combination of these or other constraints must be satisfied over every activity performed by the agent. Constraints can be arbitrary as long as they are properly encoded, interpreted, and enforced in an agent. The ROEP is the mechanism that allows an attacker to limit collateral damage and satisfy constrained objectives. Thus, a scenario maps to a state machine, an action maps to a state transition, and an ROEP maps to a set of invariants that must hold at the initial state and over every subsequent state transition.

This author knows of no attack tool that restricts itself to a subset of its behavior. That is, all attack tools thus far have had an implicit "allow all" ROEP imbedded in its internal logic.

2.5 Goal-Driven

The capability to determine, given a set of actions that *can* be performed, which one *should* be performed is what is meant by goal-driven. At any point in time, an agent may have more than one *activity* that it can perform. Knowing which activity to perform requires having an objective

and a way of linking activities into *scenarios* and evaluating which scenarios satisfy the objective (and, possibly, an ROEP). It also requires the ability to determine which of the scenarios that satisfy the objective is the most optimal.

One implementation that is feasible in a static environment is to encode pre-conditions, post-conditions, and weights for each activity as well as an objective for the agent. A scenario satisfies the objective if the pre-conditions of the scenario are initially satisfied, each intermediate activity's pre-conditions are satisfied by the previous activity's post-conditions, and the post-conditions of the scenario satisfy the objective. Whereas each activity has a weight associated with it, some function evaluates the collective weight of the scenario. The scenario with the most optimal weight is chosen for execution by the agent.

This author has never seen a publicly available attack tool that is goal-driven. However, the author has been informed that such a tool is under development at Core SDI [12], a South American security consultant firm. There have been instances of security checkers that are goal-directed [13,14,15], which could be used as planners for an attack tool.

2.6 Background Work

Until recently, most attack tools possessed at most one of these attributes. Today, tools are becoming more sophisticated and may possess more than one of these attributes. Table 1 lists attack tools or tools that model attack tools in the left column. The remaining columns in the chart indicate the presence of the attribute at the head of the column.

The following items listed under Attack Tools in Table 1 may not be familiar to the reader: Topological Vulnerability Assessment (TVA), Net Kuang, Core SDI pen testing tool, Voyager Alpha Force, and smart worms. TVA [13] is a research project at George Mason University that provides a tool set that models the vulnerabilities of a system and shows the steps an attacker can take to achieve a specified objective. Net-Kuang [14] is an instance of a Kuang System [15] that models the steps collaborating attackers can perform to achieve a particular objective. Both of these research projects focus on modeling the vulnerabilities of the system. The former also models the exploits used to attack the system. The Core SDI [12] penetration testing tool has been discussed at the DefCon Conference but not publicly released. Proponents claim it is a mobile, goal-directed attack tool that can automate penetration testing. Voyager Alpha Force is a tool that was described briefly in an online news article [16] that claims mobility and coordination. However, this author has no knowledge of its use. In the table, known capabilities are marked with an "X" and unverified claims are marked with a "?". Smart worms are discussed in Section 5.0.

Table 1. Attack Tools and Attributes

Attack Tools	Mobility	Modularity	Coordination	Policy-Constrained	Goal-Driven
<i>Ramen, 110n, Code Red, Nimda worms; Melissa virus</i>	X				
<i>Warhol, Flash worms</i>	X		X		
<i>TFN, , Stacheldracht, Trinoo</i>			X		
<i>TFN2K</i>		X	X		
<i>TVA (model)</i>	X	X			X
<i>Core SDI tool</i>	?				?
<i>Voyager Alpha Force</i>	?		?		
<i>Net-Kuang (model)</i>		X	X		X
<i>Smart Worms</i>	X	X	X	X	X

An important observation is that these fundamental attributes increase potency significantly when combined. For example, the capability of being goal-driven is best leveraged when combined with modularity and coordination. The modularity allows an attack tool to progress toward an objective without complete a priori knowledge about the environment. An objective can be formulated, scenarios can be synthesized from activities provided by peer agents, and the scenario can be executed in a coordinated way. The ability to coordinate further enhances the attacker’s ability to apply arbitrary force projection.

3.0 Potency Metrics

A rigorous discussion of attack tools requires some way to formally describe the potency of the subjects and to compare subjects. This notion of potency is arbitrary and entirely dependent on context. The context is framed by an observer who desires to know what actions a particular tool can perform and how it can perform them, and its range of targets. Answers to these and other questions collectively provide a description of the potency of the attack tool in question. Further, the determined potency of an attack tool guides the defense against it. However, identifying potency metrics is not a trivial task.

In the security community, we have seen several instances of worms over the past decade. This exposure has given us some ideas as to what makes one worm more potent than another. One worm might spread more quickly than another. Yet another worm might be able to infect a larger number of hosts. Another worm may reformat the hard drive, while another worm might just modify a Webpage to say something innocuous. How these various metrics map to a single, comprehensive metric describing total potency is, therefore, not trivial. We have also seen examples of distributed denial of service (DDoS) attacks, which has likewise given us some ideas as to what makes a DDoS attack more potent than another. Even with this exposure, however, our metrics are imprecise and incomplete.

Total potency is, therefore, not a single, comprehensive metric, but a vector of relevant metrics where each metric is unique and possibly independent of the others. For an observer to sort

attack tools based on potency, the observer must provide a function that maps the vector of potency metrics onto a sortable space.

Accepting the observation that a single metric is likely to be incomplete does not facilitate deciding which metrics are best included in the potency vector. Having experienced dozens of worms, we have some idea of what metrics are interesting when describing worms. We have fewer metrics for describing DDoS attacks, possibly because of less experience or the simplicity of the class of attack tools. The less we understand a particular type of attack tool, the harder it is to determine how to evaluate the potency of that attack tool and effectively defend against it. As the complexity of attack tools increases, the types of metrics used to describe potency are also likely to change.

Determining which potency metrics are important is prerequisite to an effective analysis of a class of attack tools. The potency metrics chosen dictate the level of abstraction needed in the attack tool attribute set discussed in Section 1.

Although the metrics of potency can be arbitrary, several metrics are commonly, albeit obtusely, used when describing contemporary attack tools. Table 2 presents several examples.

Table 2. Potency Metrics Examples

Potency Metric	Relevance
Target Set	The set of hosts that the attack tool will select as targets of an attack.
Compromise Set	The subset of the Target Set successfully attacked.
Infection Set	The subset of the Compromise Set that the attack tool actually infects.
Spread Rate	A function of the size of the Infection Set over time.
Time-To-Completion	The amount of time before the attack tool achieves its objective. This time is dependent on the type of attack carried out and may be undefined for some types of attacks. For this metric to be of any value, an (implicit) objective must exist.
Probability of Detection	The likelihood of a sensor correctly detecting an attack. The context dictates whether detection refers to observing the fact that something anomalous is occurring, correctly identifying an attack, or correctly identifying the attack tool.
Probability of Attribution	The likelihood of a sensor identifying the source of the attack given that the sensor has detected the attack, that is, $P(\text{Attribution} \mid \text{Detection})$. Context may dictate whether the attributed subject refers to a host or another entity (e.g., person, software agent).

This list of potency metrics is not intended to be exhaustive. The list is the result of an initial attempt to identify potency metrics for mobile attack tools.

To illustrate the value of evaluating the potency of threats, we consider a simplistic example. Assume we have two organizations, P and Q. Both P and Q are constantly being attacked by (the same) attackers using various attack tools, such that there is no real distinction between their situations. Both organizations have a group of IDAs who could not possibly evaluate every possible alert generated by their IDSs. Instead, each organization's IDAs tackle what they perceive as the most potent threats to their respective organizations. P uses conventional IDSs. P's IDAs investigate each alert in descending order of alert priority as determined by their IDSs. Q's IDAs, on the other hand, focus more on the attackers. They evaluate the potency metrics of the attackers, and investigate each threat in descending order based on potency.

Assume now that P and Q are both being attacked simultaneously by the same two worms, W1 and W2. W1 and W2 use different sets of exploits, although each uses exploits that trigger as high priority alerts on their IDSs. W2 propagates significantly faster than W1 and can infect a larger number of machines. P's IDAs, investigating incidents based on IDS-generated alert priority, may defend against W1 first. Q's IDAs, on the other hand, classify W2 as being more potent and therefore address defending against it first. The difference in response times can have significant impact on how much damage the victim systems incur. As the potency of attack tools increases so does the need to respond more quickly and effectively. Whereas there will never be enough time to effectively investigate every alert and every infraction, a priority-based approach will govern defensive responses. The ability to respond to the most potent attackers first will maximize resources applied to defensive response. This requires the ability to classify attackers based on relevant metrics of potency.

4.0 Mapping Attributes to Potency

Having identified potency metrics of interest and the general attributes that govern the values of the respective potency metric, a mapping must be provided that clearly defines the value of the potency metric. Whereas the attributes described in Section 1 are at a much higher level of abstraction than the potency metrics described in Section 3, further analysis of the pertinent attributes is necessary. Although it is clear that mobility is an attribute that influences the values of the Spread Rate and the Infection Set, it is not clear, at this level of description, how to map the former to the latter two.

An intermediate representation is necessary to bridge the gap. The intermediate representation must be sufficient to provide precise formulations of the desired potency metrics. The representation must also clearly span the properties associated with the original attribute. Figure 2 illustrates the relationship among the attributes, the potency relation and the intermediate representation, and the potency metrics. The notion of the potency relation includes the intermediate representation, its informal mapping to the attributes, and its formal mapping to the potency metrics.

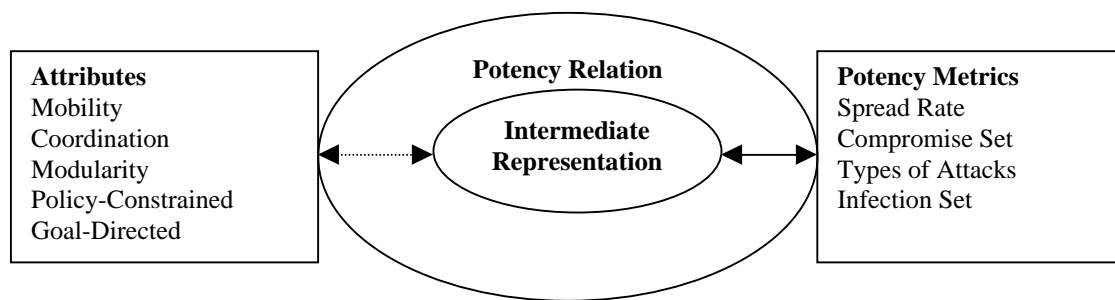


Figure 2. Mapping of Attributes to Potency

There are two criteria for the potency relation. The first is that the intermediate representation must faithfully express the attribute(s) that it represents. Precisely capturing a high-level concept is infeasible, therefore an informal mapping is the best possible. The dashed line from Attributes to Intermediate Representation denotes the fact that this mapping is performed informally. The second is that the intermediate representation must provide a way to precisely determine the

values of the potency metrics. This mapping is subject to formality and is therefore denoted by a solid line.

Ongoing research by the author is exploring a model for contemporary worms that focuses on the attribute of mobility. The model is expected to be extensible enough to incorporate the attributes of policy-constrained and goal-directed.

5.0 Smart Worms

The class of smart worms is the class of attack tools defined by {MOB, MOD, COORD, ROEP, GOAL} as described in Section 2. To better describe a smart worm, we will contrast it with a contemporary worm. Appendix A contains a sample scenario that illustrates the an attack the requires the synthesis of these attributes.

A contemporary worm propagates as much as it can. It knows a set of exploits and how to compromise and infect hosts that are vulnerable to at least one of the exploits. This process continues indefinitely: identify a target, attack the target, infect the target, repeat for every possible target host. The implicit goal of a worm is to propagate to the greatest extent possible. A question of interest is how to leverage all of those compromised machines to do something besides just propagate.

Smart worms exist to achieve a specific objective. They propagate not just for the sake of propagating, but to establish an execution environment which can be leveraged to obtain a specific objective. The smart worm has the ability to achieve that objective by infecting only the hosts that advance the smart worm's objective. Just like a contemporary worm, a smart worm has the ability to move and multiply. This allows the smart worm to assume vantage points that promote obtaining the objective. It also allows the worm to create the force projection necessary to achieve the objective. *Mobility* is the attribute that allows smart worms to set up and deploy the prerequisite force to achieve an objective.

Contemporary worms acquire the potential of a vast computational environment but fail to leverage it. Smart worms are modular and can leverage the computational environment to perform actions that will help achieve its goal. When a smart worm infects a host, it provides an execution environment for future actions. The smart worm does not need to have all possible actions imbedded in its source code if it can acquire actions as plugins and then dynamically execute them. As long as the execution environment provided is sufficiently general, any action can be encoded and executed within that environment. This generality means that any attack that can be encoded can be executed by a smart worm. *Modularity* is the key to adaptability and generality. Further, modularity refers to the ability to share and exploit knowledge of all types, both computation and data. Modular data representation allows an agent to share information about the environment gleaned from reconnaissance activities as well as tools that can be used to manipulate the environment.

Some contemporary worms have shown that they can use static coordination in determining how they spread. For example, each infected host might first look to spread locally (possibly by attacking hosts with the same first three octets as the local host's IP address) before spreading globally (attacking a host with a more significant octet modified). Recently, researchers have

suggested improvements to this coordination [4,5]. However, these improvements are still statically defined and are limited to a priori knowledge about the environment. Although coordination while spreading is important, coordinating activities after the appropriate force has been deployed allows more flexibility in the types of attacks that can be executed.

A smart worm coordinates not just while spreading but after the infection phase is complete. The ability to coordinate activities leverages the advantages gained by being modular. Although some objectives may be achievable from a single agent's current vantage point using actions currently known, other objectives may require an increase in knowledge or force projection.

This increase in knowledge may materialize in the form of new actions or new information about the environment. An agent can coordinate with other agents and acquire the actions that will successfully achieve an objective. An agent can learn about another agent's knowledge about the environment without having to perform the reconnaissance itself. Alternatively, an agent may need an increase in force projection from peer agents for an activity to be successful.

Coordination is the key to leveraging information superiority and effective force projection.

Given a set of known actions, it does not always make sense to perform all of those actions on every possible target all of the time. Discretion, in many cases, is to be preferred just as there are cases where smart bombs are preferred to contemporary bombs. Contemporary worms are entirely indiscrete. They do all that they can whenever they can against everybody they can. On the other hand, a smart worm performs only the actions that it is allowed to perform. This self-restraint is enforced by an ROEP. Every action an agent performs must be explicitly approved or rejected by the ROEP. The content of the ROEP is critical to how a smart worm behaves.

The ROEP contains criteria that are used to evaluate every action that an agent performs throughout all phases of activity (e.g., performing reconnaissance, attacking, communicating, infecting). Arbitrary constraints can be defined in the ROEP and enforced by the agent if metrics are associated with the actions that can be used to evaluate compliance. Thus, the ability to be *policy-constrained* allows a smart worm to be as selective and precise as necessary while accomplishing an objective.

Given a set of actions that may be performed, a smart worm needs to decide which action should be performed. Unlike contemporary worms that do everything they can, a smart worm does only what needs to be done to achieve an objective. A smart worm is instantiated with an objective. The smart worm evaluates its initial condition and inventories its set of known actions. It then identifies scenarios that will achieve the objective and executes them accordingly. The ability to be *goal-directed* enables a smart worm to back chain from an objective and identify a successful course of action that will achieve the objective.

A smart worm is instantiated in a specific environment with three parameters: an objective, an ROEP, and a knowledge base (KB). The KB includes knowledge about exploits, propagation mechanisms, payloads with their respective interfaces, and the environment. The smart worm identifies what it needs to know and do to achieve the objective. A set of scenarios is constructed out from known actions (those in the KB) where each scenario achieves the objective and satisfies the ROEP. If the set is empty, then the smart worm cannot achieve the objective and

returns a failure. Otherwise, the smart worm executes the scenario over the environment. Figure 3 illustrates the process a smart worm goes through before performing an action.

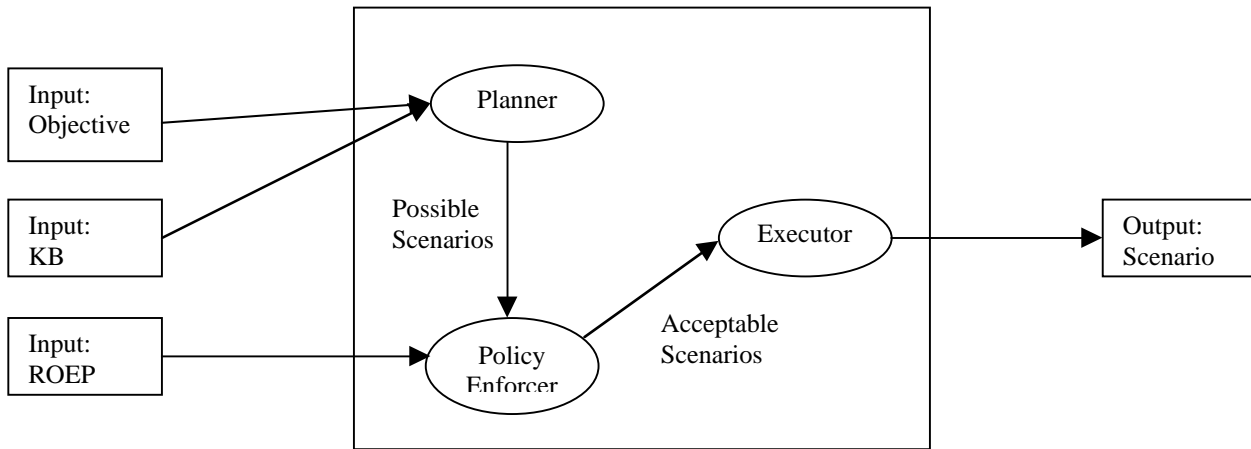


Figure 3. Smart Worm Process Flow

6.0 Conclusions

This paper presents a set of attributes that map a space of classes of attack tools. The ROEP is a novel attribute that is shown to allow an attacker significant flexibility and precision in attacks. The class of smart worms, the threat posed in this paper, is likely to appear in the next few years. A methodology is presented that can be used to evaluate the potency of this and other classes of attack tools. This methodology can be used to achieve two purposes. The first is predictive: given a hypothetical threat with specified attributes, we can identify relevant metrics of potency and the means to evaluate their values. The second is defensive. The values that govern the values of the potency metrics are the very factors that a defender has to be able to influence in order to alter the potency of the attack tool. Also, having evaluated the potency of attack tools engaged in current attacks, a defender can better allocate defensive resources. The methodology, therefore, helps a defender evaluate and combat against both hypothetical and practical threats.

Current work in progress by the author is applying this methodology to contemporary worms (including the attributes of mobility, modularity, policy-constrained, and goal-directed). Further work is needed to enhance the model under development to include the attribute of coordination.

Appendix A

The following scenario illustrates how a smart worm is instantiated and how it can carry out an attack. Figure 5 shows the environment as it is given to the smart worm during instantiation. For simplicity, the smart worm possesses a representation of the connectivity, sensor capabilities, and vulnerabilities, except for **V**, as shown in Figure 5 all in its KB. The smart worm does not know the vulnerabilities of **V**. A smart worm agent is instantiated on **A** and is henceforth referred to as **A**. **A** is instantiated with two objectives: (1) infect **V** and (2) cause **V** to initiate a Transmission Control Protocol (TCP) connection to **A** and await further commands. **A** is instantiated with the three constraints in the ROEP: (1) two hosts must be infected serially from **A**'s perspective before directly attacking **V**, (2) exploits that can be seen by Intrusion Detection System 2 (**IDS2**) must be executed while another host is performing a FLAKattack on **IDS2**, and (3) no exploit is to be sent past **IDS3**.

A's KB also includes exploits and other actions. UDPattack (an attack on a User-Datagram-Protocol-based service), TCPattack1 and TCPattack2 (attacks on Transmission-Control-Protocol-based services), and ARPattack (an attack on the Address Resolution Protocol) are known exploits. TCPpropagate and UDPpropagate are known propagation mechanisms. FLAKattack (a diversionary attack) is another known attack, but is not classified as an exploit. The nodes labeled **FWi** are firewalls. Both firewalls prevent any external host from initiating a TCP connection to an internal host; all other traffic is allowed.

B and **I** are known to be vulnerable to UDPattack. **B**, **I**, and **K** are known to be vulnerable to TCPattack1. **C** and **H** are known to be vulnerable to TCPattack2. **D** is known to be vulnerable to ARPattack. **V** is also vulnerable to ARPattack, but this is not initially known to **A**. All of the exploits contain bootstrap code which initiates a propagation mechanism. All IDSs sensors are vulnerable to FLAKattack.

This sample scenario illustrates how each of the attributes can be used to enhance the potency of the smart worm. The fact that the smart worm is mobile allows it to leverage various vantage points to accomplish its objective. The ability to coordinate attacks (i.e., the FLAKattack and TCPattack1 of step 3) allows the agents to execute attacks that are more sophisticated and covert. The ability to share information (i.e., the ARPattack exploit) allows the agents to decrease the size of the statically linked payload. Further, the worm agent is not required to know what would be needed to attack and infect **V** until that reconnaissance is performed. The ability to be goal-directed frees the attacker from needing to follow every step explicitly before hand. All **A** needs to do is propagate the objectives, ROEP, and KB, and then wait for **V** to initiate a TCP connection.

The ROEP keeps each of the agents from performing actions that were previously deemed unacceptable. This preserves some measure of covertness to the activities. For example, the shortest path with respect to capabilities leads **A** to attack **H** with TCPattack2. There is no intermediate IDS sensor between **H** and **V**, so FLAKattack is not needed. This scenario is discarded only because it fails to satisfy the ROEP although it is less complicated. The leapfrog tactic required by the ROEP also makes attribution more difficult. In addition, discriminating between links can be helpful in avoiding detection.

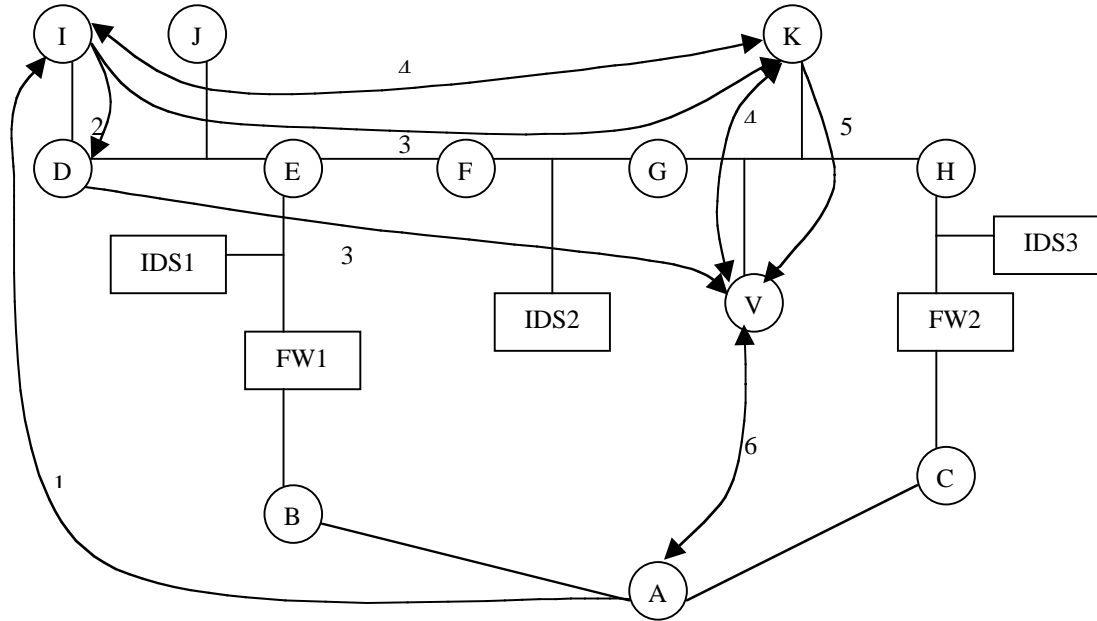
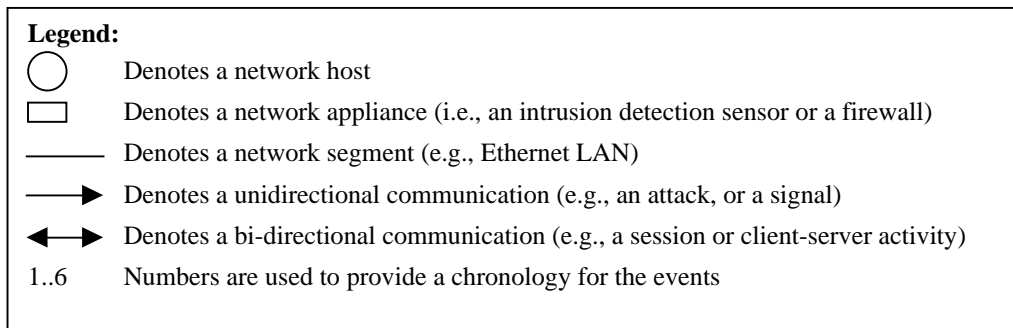


Figure 5. Sample Smart Worm Scenario



The attack proceeds at a high level as follows:

0. A is initialized with an objective, an ROEP, and a KB (not represented in Figure 5).
 - Objectives:
 - Infect V.
 - Cause V to initiate a TCP connection to A and await further commands.
 - ROEP:
 - Two hosts must be infected serially from A's perspective before directly attacking V.
 - Exploits that can be seen by IDS2 must be executed while another host is performing a FLAKattack on IDS2.
 - No exploit is to be sent past IDS3.
 - KB:
 - Attacks: UDPattack, TCPattack1, TCPattack2, ARPattack, and FLAKattack
 - Propagation Mechanisms: TCPpropagate and UDPpropagate
 - Network Topology
1. A attacks I with a UDPattack and uses UDPpropagate to infect I.
 - A gives I the following objectives, ROEP, and KB:
 - Objectives:

- Infect **V**.
 - Cause **V** to initiate a TCP connection to **A** and await further commands.
 - ROEP:
 - One hosts must be infected serially from **I**'s perspective before directly attacking **V**.
 - Exploits that can be seen by **IDS2** must be executed while another host is performing a FLAKattack on **IDS2**.
 - No exploit is to be sent past **IDS3**.
 - KB:
 - Attacks: UDPattack, TCPattack1, TCPattack2, and FLAKattack
 - Propagation Mechanisms: TCPpropagate and UDPpropagate
 - Network Topology
2. **I** attacks **D** with ARPattack and uses TCPpropagate to infect D.
 - **I** gives **D** the following objectives, ROEP, and KB:
 - Objectives:
 - Execute FLAKattack against **IDS2** for 20 seconds after a signal from **I** and then clean up after self (i.e., remove all source files, remove all temporary files, clean the logs, and terminate the program).
 - ROEP:
 - No other attacks or events may be performed.
 - KB:
 - Attacks: FLAKattack
 - Network Topology
 3. **I** signals **D** to begin FLAKattack (and **D** begins the attack).
 - **I** attacks **K** with TCPattack1 and uses TCPpropagate to infect K.
 - **D** cleans up and terminates.
 - **I** gives **K** the following objectives, ROEP, and KB:
 - Objectives:
 - Infect **V**.
 - Cause **V** to initiate TCP connection to **A** and wait for more commands.
 - ROEP:
 - No exploit is to be sent past **IDS3**.
 - Any exploit sent past **IDS2** must be performed during a FLAKattack to **IDS2** from another host.
 - KB:
 - Propagation Mechanisms: TCPpropagate and UDPpropagate
 - Network Topology
 4. **K** scans **V** and determines that **V** is vulnerable to ARPattack and updates its Network Topology.
 - **K** requests and then receives ARPattack from **I**.
 5. **K** attacks **V** with ARPattack.
 - **K** infects **V** with TCPpropagate.
 - **K** gives **V** the following objectives, ROEP, and KB:
 - Objectives:
 - Initiate TCP connection to **A** and wait for more commands.
 - ROEP:
 - No exploit is to be sent past **IDS3**.
 - Any exploit sent past **IDS2** must be during a FLAKattack on **IDS2** from another host.
 - KB:
 - Network Topology
 6. **V** initiates a TCP connection with **A**.

8.0 References

- [1] Hupp, Soch. The 'Worm' Programs—Early Experiences with a Distributed Computation. In CACM 25(3) pp. 172-180. March 1982.
- [2] Fred Cohen. A Formal Definition of Computer Worms and Some Related Results. Computers & Security, Volume 11, 1992, pp. 641-652.
- [3] Fred Cohen. Computer Viruses – Theory and Experiments. Computers & Security, Volume 6, 1987, pp.22-35.
- [4] N. Weaver. Warhol Worms: The Potential for Very Fast Internet Plagues. <http://www.cs.berkeley.edu/~nweaver/warhol.html>
- [5] Staniford, Grim, Jonkman. Flash Worms: Thirty Seconds to Infect the Internet. Silicon Defense, August 16, 2001. <http://www.silicondefense.com/flash/>
- [6] Cohen. A Note on Distributed Coordinated Attacks. April 1996. <http://all.net/>
- [7] Steve White. Open Problems in Computer Virus Research. Presented at Virus Bulletin Conference, Munich, Germany, October, 1998.
- [8] Jeffrey Kephart, Steve White. Measuring and Modeling Computer Virus Prevalence. In Proceedings of the 1999 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, California, May 24-25, 1993, pp.2-14.
- [9] Hayes. The Year of the Worm. Security Focus. August 20, 2001. <http://www.securityfocus.com/>
- [10] Jason Barlow, Woody Thrower. TFN2K - An Analysis. Axent Security Team. February 10, 2000. http://security.royans.net/info/posts/bugtraq_ddos2.shtml
- [11] <http://www.nessus.org>
- [12] I. Arce, M. Caceres. Automating Penetration Tests: A new challenge for the IS industry? Core SDI. Presented at DefCon in Las Vegas, July 2001.
- [13] Ronald Ritchey, Paul Ammann. Using Model Checking To Analyze Network Vulnerabilities. In Proceedings 2000 IEEE Symposium on Security and Privacy.
- [14] D. Zerkle, K. Levitt. "NetKuang – A Multi-Host Configuration Vulnerability Checker." In Proceedings of the Sixth USENIX Unix Security Symposium, San Jose, CA 1996.
- [15] R. Baldwin. Kuang: Rule-Based Security Checking. Ph.D. Dissertation. Laboratory for Computer Science. Massachusetts Institute of Technology. MIT/LCS/TR-401. March 1988.
- [16] W. McAuliffe. Hybrid DDoS Worm Strikes Microsoft SQL Server. ZDNet, 23 November 2001. <http://news.zdnet.co.uk/story/0,,t269-s2099780,00.html>

Approved for Public Release; Distribution Unlimited
© 2002_The MITRE Corporation. ALL RIGHTS RESERVED.