
# *Nematodes – Beneficial Worms*

V 1.0
Original: September 2005

Dave Aitel
dave@immunitysec.com

# Who am I?

- NSA->@stake->Immunity
- Currently Researcher: Immunity, Inc.
  - Consulting (product assessments)
  - Immunity CANVAS
  - Immunity Partner's Program
  - Training
  - Ongoing research in exploits and software vulnerabilities

# What is a worm?

- Any self propelled replicating program
- Commonly used for nefarious purposes such as overloading the Internet and inhibiting the valid distribution of porn

# What is a Nematode

- A controlled worm that can be used for beneficial purposes
  - Making your network self protecting!
- “Nematode” is a phylum of primitive worm-like organisms often used to get rid of other pests

# Hypothesis

- In the near future, many organizations will use nematodes to lower the costs of securing their networks
  - ISPs
  - Governments
  - Large companies
- ROI will overcome unreasonable fear

# Agenda

- Reasons to create a nematode
- Protocols for controlling your nematode
- Automatically creating your nematode from vulnerability information
- Other uses for nematodes

# Why create a worm?

- I want to secure my network. Today, this is very expensive!
- Hard problems, like security, require novel and difficult approaches, like controlled worms
- Other hard problems are also solvable with worms
  - Distributed searching
  - Systems management on a large scale

# Networks are a jungle, not a tundra

- Complex, dynamic network architectures are the standard
- These often evolve from simple flat networks as a company grows
- Networks are not documented – asset management is an expensive problem to solve
- Current defenses are still weak and expensive!
- Dream: But what if my network was self discovering, without the need to install monitoring stations all over the place?

# Nessus (or similar) scanners as asset management tools

- Right now, nessus-like scanner stations are dotted all over your network landscape, peering into the unknown like telescopes into a dark night
- As soon as you have finished your Nessus-like scan, it is out of date and you must begin again
- Nessus-like modules may generate false positives
- Exploits can be written nearly as quickly as Nessus signatures!

# Network Segmentation

- Any network segmentation adds to the costs of a solution that requires direct visibility across the network
- It is hard to get our scanners close to our targets
- Machines that pop in and out of the network remain a false negative issue and are commonly cited as problem vectors

# Other potential nematode features

- Searching entire network, without regard to network architecture
  - Worms make great filters: I want the latest sales spreadsheet that anyone on my team has done! Go get it.
- Moving intelligence across the network

# Exploits vs. Worms

- A worm does not need an exploit
  - testvuln1.exe worm is a good replacement for any installed management agent
    - Minus authentication, of course.
  - Even very polished exploits fail sometimes
- Some exploits may be very difficult to write a worm for!
  - These are more rare than people like to pretend

# Establishing legal mandate

- Mandate to attack a machine differs from exploits to nematodes
  - This is the largest part of the “fear factor”
- There are plenty of places where running exploits is perfectly legal and desirable
  - Your own network
  - Someone else's where you have permission
    - “Penetration testing”

# Exploits have easy mandates

- Reasonable knowledge where your target is (it is on the same network you are allowed to attack, for example)
- Slow scale of penetration allows for manual verification at every step
- Mistakes **do happen**, but are generally of low consequence due to human interaction
- Logging is easy to do

# Nematodes have to work harder to establish clear mandate

- Rely on outside indicators to find out where they are running
- Rely on outside indicators to find out where they are allowed to attack
- Logging is quite difficult (distributed problem)

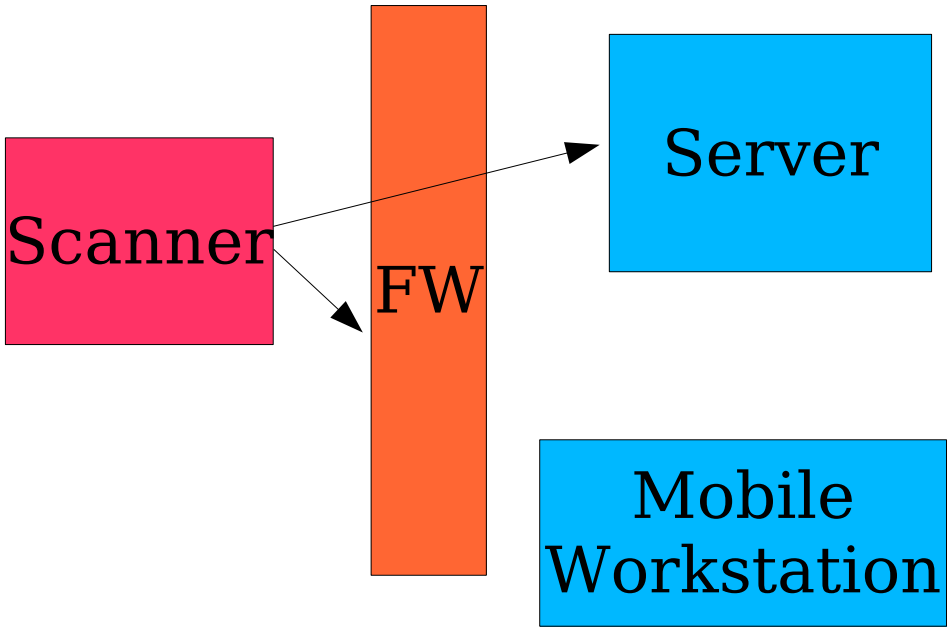
# Halfway point: Exploit scanners

- For ip in range(192.168.1.0,192.168.1.255):
  - ret=exploit(ip)
  - If ret:
    - ret.patch()
    - ret.reboot()

# Scanners are not perfect...

- Scanners and automated exploit technology save money by solving the asset management problem
- You have a clear mandate on your own network!
- Can have large and complex support structures
  - Specialized recon tools
  - large shellcode is possible

# Scanner Problems




- Multiple networks require multiple scanners
  - Administrators now have an impetus to avoid network segmentation :<
- Scanners absorb bandwidth for discovery
  - (Even with a switch's help)
- Hosts are constantly popping up (time disparities)

# The solution: Nematodes

- Every host is a scanner
- Every host can generate scanners by automatically deploying nematodes
- Hosts that are secure or unreachable, are not a problem!
- Scans that are not relevant just die out

# Problems with nematodes

- Worm are really hard to write
- Worms also use large amounts of network bandwidth
  - Need smart algorithms to counteract this
    - But smart algorithms make for very large worms!
- Worms are harder to target and control – fear factor ensures
  - Need to ensure legal access

# Validating mandate on a nematode's target

- Is target on a white-listed network
  - Hard to do with private address spaces
- 2 factor authentication method
- Does target run our custom worm management agent ?
  - Friend/Foe system

# Nematokens

- Nematoken server should only respond to requests from networks we are allowed to attack from/to
  - Ex: Does a192.168.1.2.mytokenserver.com exist?
    - Yes: ok to attack it.
- DNS is a good one, but there are plenty of other options

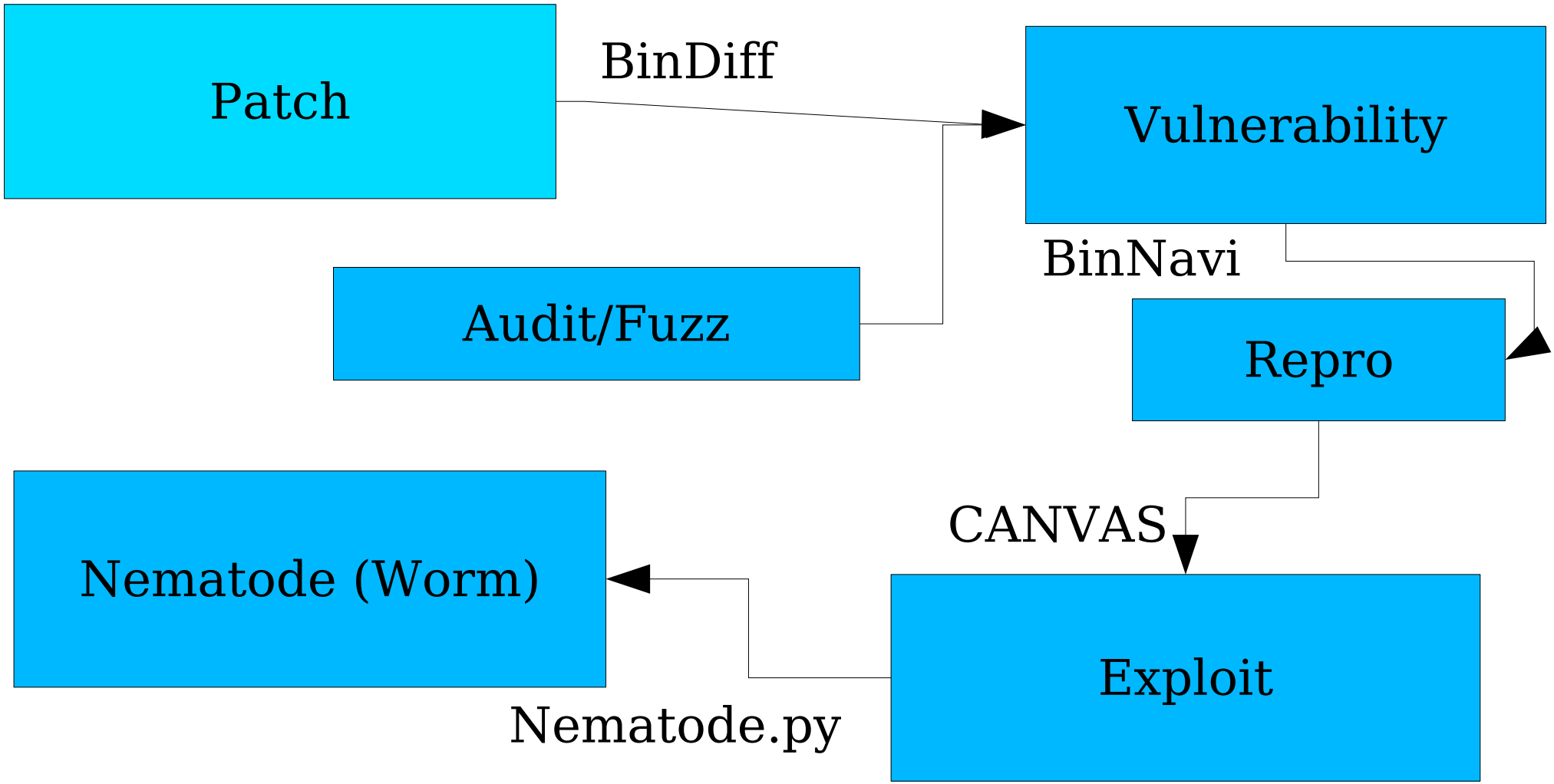
# Nematode Implementation

- Every nematode implementation may differ
- Immunity's goal is to make our nematode implementation flexible, such that it can be deployed on the fly
  - Something dynamically created but reliably controllable
  - Operates entirely in memory
  - single shot (no callbacks)

# Automatically Generating Beneficial Worms

- How do we get from vulnerabilities to usable beneficial worms?
  - Vulnerability
  - Python Exploit
  - Nematode Intermediate Language
  - Nematode Test Framework
  - Nematode payload deployed

# Our Problem Space



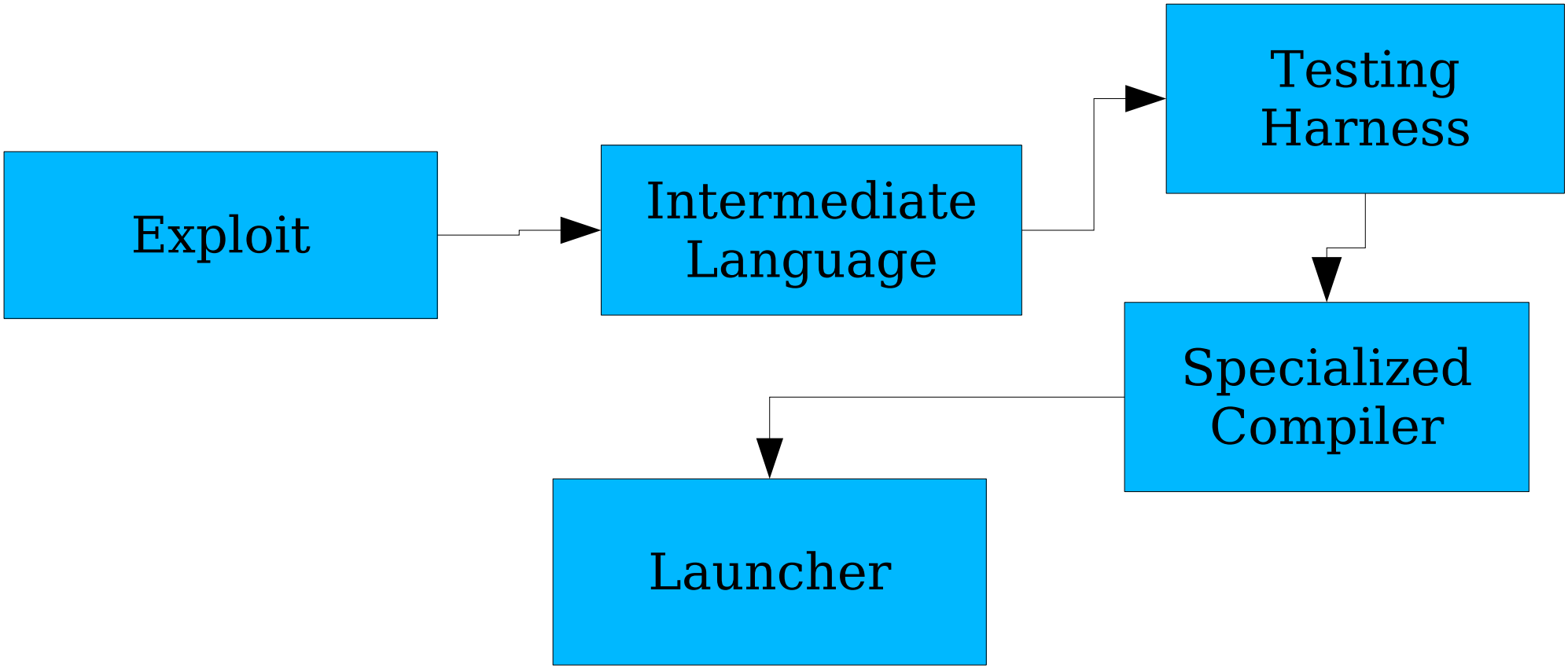
# Exploit frameworks are good for nematode development

- Ideal framework has following characteristics
  - Completely written in Python
    - Including many network protocol libraries
  - Many exploits
  - Exploits are written to an API, rather than haphazardly
  - Built-in assembler and compiler!

# Example Exploit in Python

- Class with simple member functions
  - Makesploit() - Creates the buffer to send to the target
  - Nops() - generates nops
  - Stroverwrite() - Python version of memcpy
  - connect_to_host() - Connects to the target
  - Runonce() -Sends the buffer

# Our Nematode Work-flow



# Nematode Intermediate Language (NIL)

- Specialized and simplified “assembly for worms”
- Useful for converting exploits into Nematodes quickly and easily
- Exploits can be written to NIL directly
  - This is probably not a good idea, but for complex worms hand-modification may be necessary

# Automatically Generating NIL

- Python is introspective/reflective
  - We simply override our internal API to generate a NIL file instead of running the attack!
- exploit.nops=self.nops
- exploit.run()

# ./nematode.py demosploit

- As simple as running one python script which loads the module, replaces the functions, and calls runonce()
- <see amazing demo now>

# Demosploit -> NIL

```
nops 5000
stroverwrite %B8%DD%FF%BF 1036
stroverwrite %CCtheshellcode 800
startloop
connect_random_host 5151
sendall
closesock
endloop
```

Ooh, ahhhh...

# NIL -> Test phase

- neminterp.py will interpret NIL as an aid to testing
- If the exploit still works, we're good to go

# Building our final payload

- Go from NIL to assembly language
  - This requires a specialized NIL compiler with hand written assembly
- Inject assembly language into test framework
- Watch it go!

# Demos are fun

- <See amazing assembly code now>
- <See amazing worm demonstration now>

# NIL Assembler

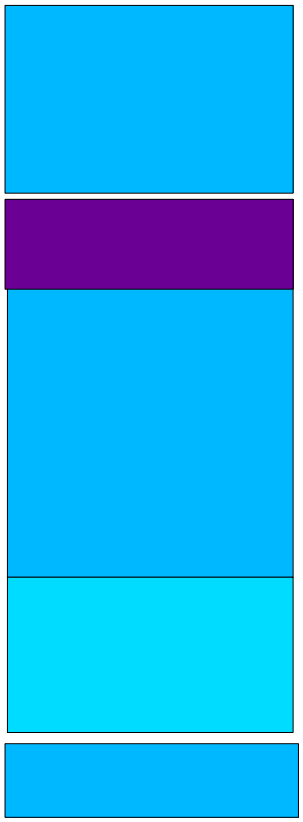
- Want to minimize space
- Need a unified function table
- Need to avoid badcharacters
- Also need to be flexible

```
nem=nem_linux_X86()
nem.addAttr("nem_prelude",None)
nem.addAttr("nops",[5000])
nem.addAttr("stroverwrite",["%27%83%
nem.addAttr("stroverwrite",[shellcodes
nem.addAttr("startloop",None)
nem.addAttr("connect_random_host",[5
nem.addAttr("sendall",None)
nem.addAttr("closesock",None)
nem.addAttr("endloop",None)
data=nem.get()
```

# Results for Nematode v0.1

- Linux demosploit (no bad chars)
  - <5 minutes from exploit to worm
  - Worm is <280 bytes
    - Currently no real payload other than replication
      - Just like most worms!
  - Simple incrementing scanner
    - Will use /dev/urandom in v0.2

# Nematode Assembler Future Features




- Select from multiple decoder/body parts to account for different bad character lists
  - Or use automatic assembler heuristics

# Future nematode problems

- MSRPC creates interesting issues with regards to constructing our attack string
  - Ideally we'd use native API
    - This requires a working minimized typedef library
      - Which is totally doable.
- Doing multi-stage worms is also possible, but less reliable due to NAT

# Witty made people cry

- Worm shocked people with rapid (48h) deployment after announcement
- Witty is nearly identical to this sort of simple stack overflow bug
  - Need to reconsider whether worm creation tools are already in wide use...
  - A good attacker can reliably create a worm that appears before your half-baked IDS signature does

# Worms can be fast

- Signature based worm protection is only useful as a diagnostic, not as a prophylactic
- Some interesting work has been done in automatically detecting worm signatures
  - Polygraph: Automatically Generating Signatures for Polymorphic Worms, James Newsome, Brad Karp, and Dawn Song.

# But not all worms are Nematodes – how do we control this thing?

- We can now dynamically and quickly create worms – now what?
- Controllable worms (nematodes) need
  - State
  - Payload
  - More complex network protocols

# Adding state to the equation

- Append state section to header or footer of payload
  - Need to encode it, potentially
- Have nematode body modify state section before sending off to next target

# Attacking In Scope Networks Only

- Options: Nematoken/Whitelist
  - Need not a whitelist, but a whitegraph!
- 192.168.1.* is fair game
  - But only if attacked from 192.68.2.*
- Simple graph walking algorithm is necessary – in shellcode

# Whitegraph implementation

- Need to store where we came from, and match against the graph to determine where we are allowed to go next
  - Need wildcards in the graph
  - This is a complex parsing problem to have to do in shellcode
- Also potentially quite useful for avoiding network telescopes

# Payload

- Payload may be
  - Install management agent
  - Install patch and reboot
  - Report to central server
  - Whatever you can think up
- Dynamically mix and match!

# Conclusion

- Solving the “Am I allowed to attack this” problem is not impossible for a nematode system
- Frameworked exploits are essential to do automated development of nematodes
- Generalized technique is completely cross platform

# Resources

- Worm Blog run by Jose Nazario
- WORM 2005 academic conference at George Mason
- Journal of Computer Virology

# Questions?

- Did we answer more than we asked?