# Rule Verification + UI for NPF on NetBSD

Sanjay Chandrasekaran

# NetBSD (fun facts mostly)

- Unix-like Operating system based off Berkeley Software Distribution (BSD)
- Forked from 386BSD
- Kernel-Space Scripting w/ .lua files (relatively new, doesn’t seem appealing)
- Key Features:
  - *Portability* – uses a hardware abstraction layer so device drivers can have machine independent components that are reused over many platforms
  - *Symmetric Multiprocessing* – has optimized for everything except network protocols and most device drivers
  - *Security* – supports many features for exploit mitigation, authorization etc.

# NPF (less fun facts)

- Layer 3 Filter that intercepts packets and performs IP reassembly
- Has extension framework for adding custom modules
  - Packet Logging
  - Traffic Normalization
  - Random Blocking
- Often leaves users with misconfigurations in their firewalls

- Just kidding, but accidents can happen!

# Project Additions

- WebUI interface for NPF using Ur/Web
  - Packaged and integrated with NetBSD
- Verification of Firewall Rules
  - Verify the correctness of rules given a list of properties
  - Run whenever rules are added/removed
  - Verification of implementation

# Resources

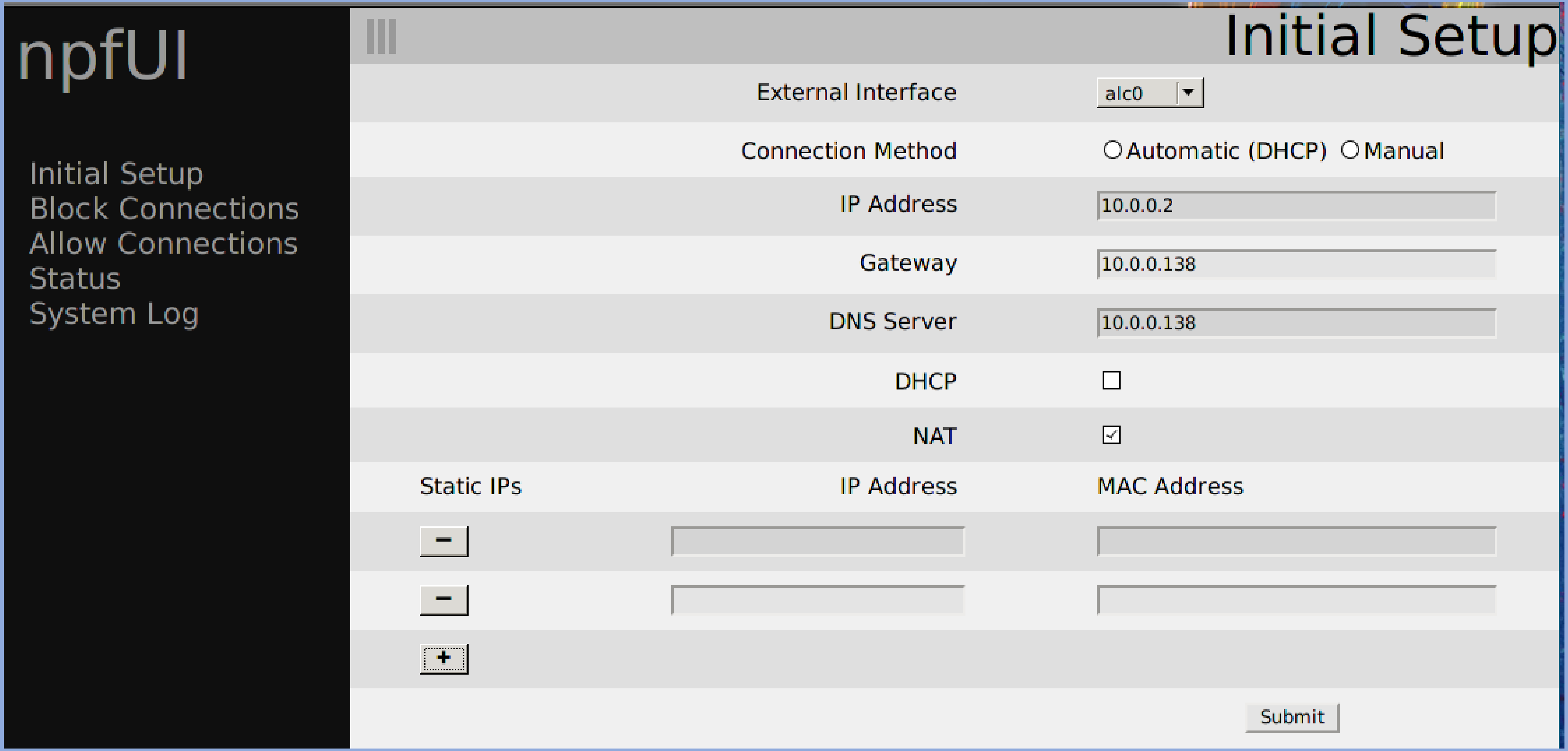
- Vagrant running NetBSD
  - Potentially Virtualbox instead
- Machine for running/hosting the Proof Generator (preferably w/ decent processing power)
  - Linux Machine
  - 

# Code Base

- User Control (npfctl) ~ 2000 lines of code
- NPF Kernel Component ~ 3000 lines of code

- Will mainly use these to understand functionality of the firewall
  - Look for possible ways to install/extract firewall rules

# Creating a UI for NPF



# Writing the UI in Ur/Web

- Ur/Web features
  - Does not return invalid HTML
  - Does not have mismatches between HTML forms and the fields expected by their handlers
  - Compiler produces efficient object code w/out garbage collection
  - Easier Implementation Verification
- UI Features
  - Setting Interface Addresses and Routes + Configure DHCP,
  - Manage firewall rules, allowed/blocked connections
  - NAT (including port redirection) + Wireless AP

# Creating a Language for Firewall Rules

- Rule format
  - pass stateful in final family inet4 proto tcp flags S/SA from $source port $sport to $dest port $dport apply “someproc”
- Property format
  - [ $interface , ( $source , $sport , $dest , $dport , $proto ) ] → action
- Connectives
  - first match / last match
  - may need 1-2 new connectives

# Automated Theorem Proving (ATP)

- Sophisticated Constraint Solver
  - ATP competition (CASC) every year, Vampire won many times
- Takes in a set of Axioms along with the Theorem we want to prove
  - The theorem will be a proposition that we create from the set of firewall rules
  - The axioms will be the set of true facts
  - The proving of the rules will most likely be offloaded from the router/NetBSD but a proof will be returned which the router should have a verifier for
  - If the verifier turns out to be too complex to put into NPF, it can also be offloaded and the rules can be signed with a cert verifying it's correctness
  - Propositions with up to 30-40 connectives can still be proved in a reasonable amount of time, quantifiers are the main time kill

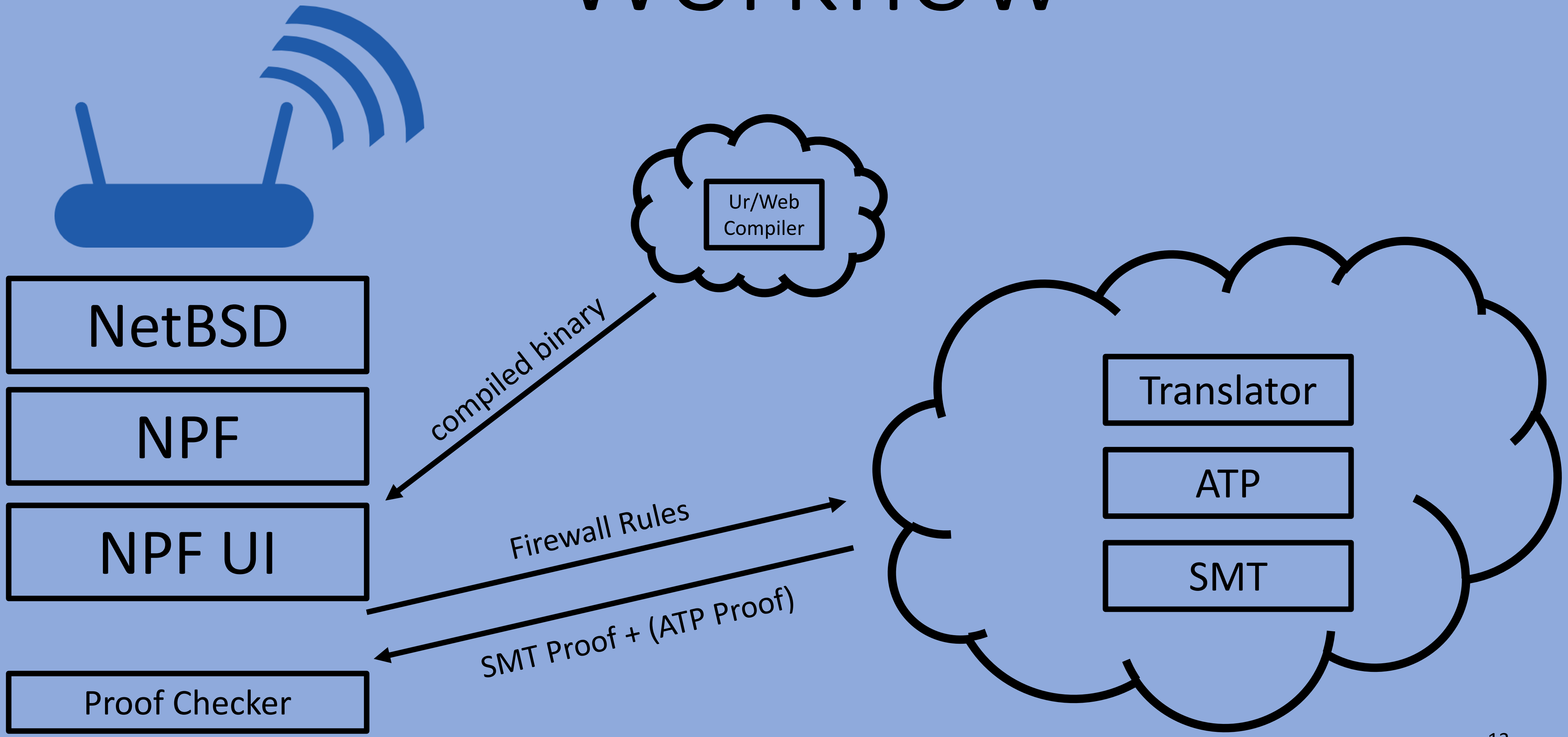
# Satisfiability Modulo Theory (SMT)

- Specific kind of ATP that takes advantage of linear arithmetic theories
- Can represent IPs as numbers in this logic rather than decomposing the IP to the bit level
- We will negate the proposition and try to prove it with an SMT solver such as Z3
  - If the negated proposition is unsatisfiable then the SMT will output a proof as to why original proposition is true
  - If the negated proposition is satisfiable then the SMT will output a configuration for which the negated proposition is true, which will be used as a refutation for the original proposition

# Documentation

- NPF
  - Good man pages, functionality pretty well-documented
- Ur/Web
  - All Functionality Documented in Manual
  - Some tutorials/examples
- ATP / SMT
  - Many Research Papers and Docs
  - Ruben Martins & Frank Pfenning, Matt Frederikson & Jean Yang ☺
- NPF UI
  - Will need to create a User Manual for the UI
- Proof Generator/Verifier
  - Will need to create dev doc in case extra functionality is to be added

# Workflow



NetBSD
NPF
NPF UI
Proof Checker
Ur/Web Compiler
compiled binary
Firewall Rules
SMT Proof + (ATP Proof)
Translator
ATP
SMT

# Lines of Code/Proofs to Write

- NPF UI ~ 500 lines
- Translator ~ 100 lines
- ATP/SMT ~ 100 lines
- Verification of Implementation ~ *a few*
- Correctness Proofs by hand ~ *a lot*

# Tentative Schedule

- 10 weeks remaining this semester
- **Stage I** (1-2 weeks *currently happening*)
  - Understand how ATP works, and how the implementation would change based on logic
  - Plan out UI along with which features will/will not be included
- **Stage II** (4-5 weeks) ~ Nov 6
  - Write UI for NPF, along with translator, be able to get an output of the currently added rules; hopefully get pull request
  - Create a Language for the Firewall Rules, attempt correctness proofs by hand
  - Plan out what will need to be implemented for ATP
- **Stage III** (4-5 weeks) ~ Dec 4
  - Fix any UI issues, do proofs by hand verifying implementation of UI
  - Implement ATP / SMT
  - Create a verifier for a proof that is output to accompany the NPF UI
  - Add ATP as a service for UI
- **Stage IV** (1-2 weeks) ~ Dec 15
  - (test $\wedge$ debug) U (finish)          [hopefully this is all the temporal logic I'll have to write ☺]

# Issues/Challenges

- ATP can be complex in runtime, the end goal should be something that is reasonable to run anytime a rule is added
- The verification of the proofs in the UI may need to be replaced by certificates, in which case it would probably go unused
- Modeling Higher Level Properties such as Information Flow are tough and will take more time

# Questions?

- Discussion
- Advice
- Thoughts

happy friday ☺