Network Security

Spring 2010, Assignment 2

Shellcode Detection

Chiung-Yi Tseng

Bruno Coutinho

Part1:

Heuristic 1: Sled detection

The false positvie rate of heuristic 1 keeps 0% at all MIN_SLED.

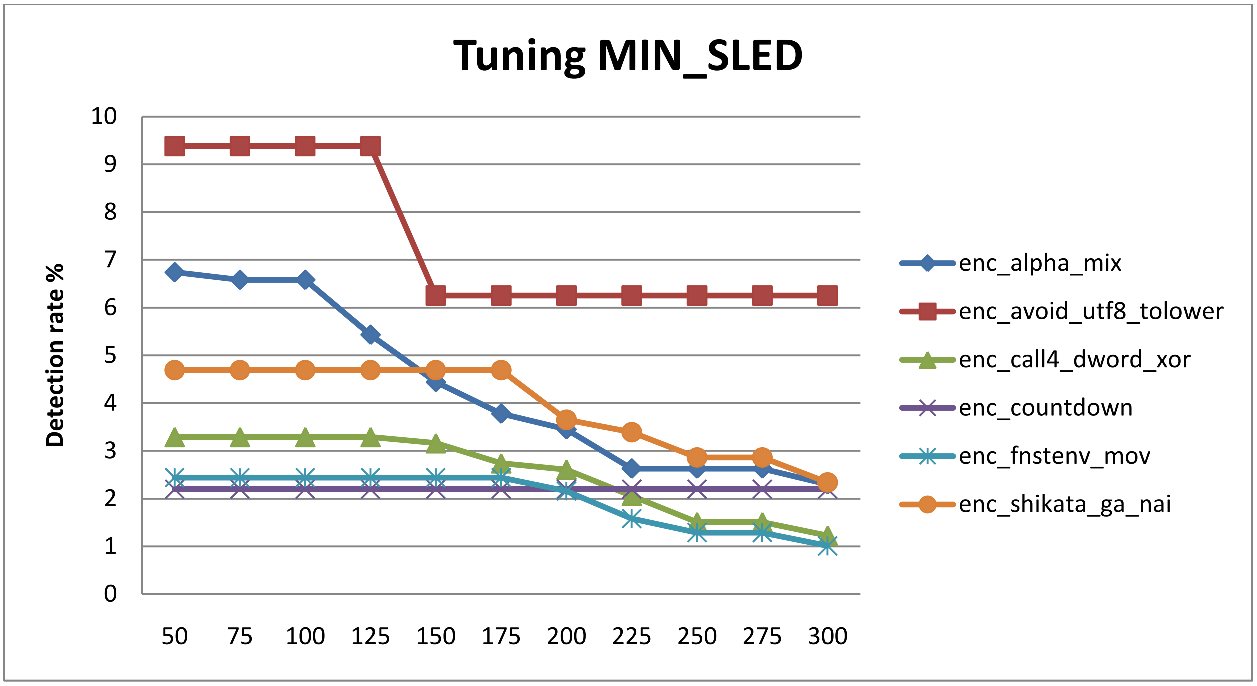


Figure 1: tunig MIN_SLED

The detection rate of Sled detection is very low. We try to reduce the MIN_SLED to 10, the detection rate of enc_avoid_utf8_tolower increases to 9.38%, enc_alpha to 6.74%, while the false positive rate remains 0. The performance of sled detection is slightly better at detecting enc_avoid_utf8_tolower, while remains nearly the same at detecting enc_alpha. The possible reason is that enc_avoid_utf8_tolower is non-self-contained encoder.

To fully utilize sled detection, one may assign a small value (~10) to MIN_SLED, although using sled detection solely is not a good idea.

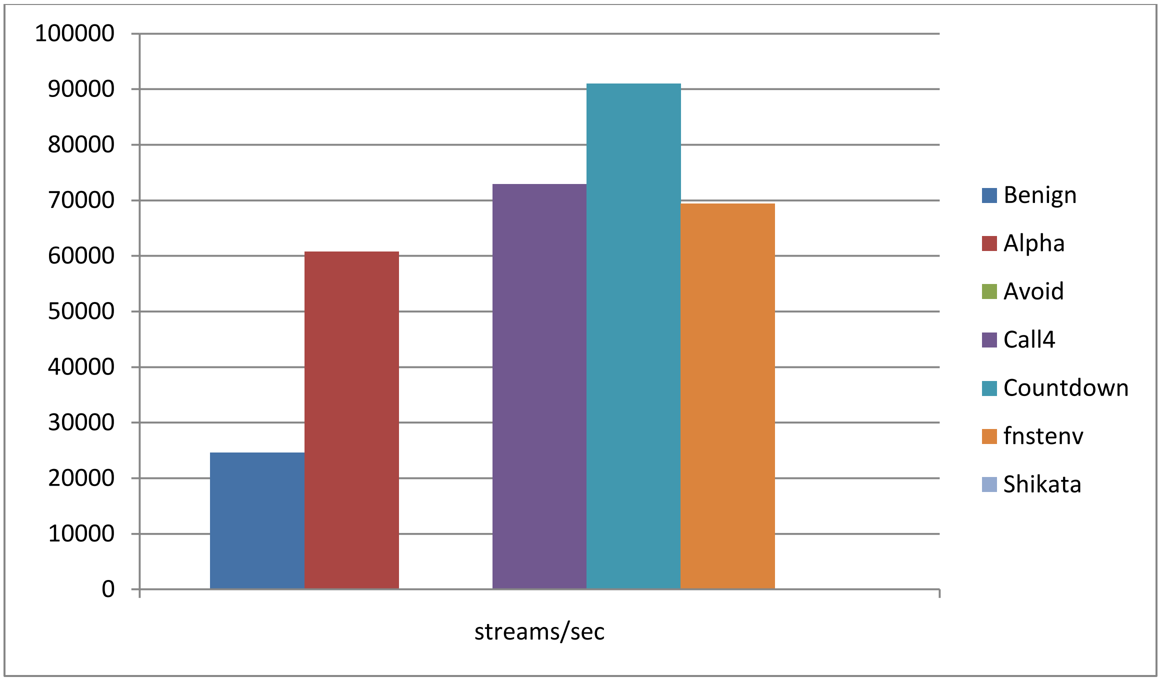


Figure 2: MIN_SLED = 10

	Benign	Alpha	Avoid	Call4	Countdown	fnstenv	Shikata
streams/sec	24644.382	60800		72900	91000	69400	
sec	1.78	0.01	0	0.01	0.001	0.01	0
streams	43867	608	32	729	91	694	384

* Note that the since the number of streams of avoid _utf8_tolower and Shikata_ga_nai are very few, the execution time is too quick. We can not get steams/sec data.

Among all encoders, the throughput of Alpha is the highest, while the throughput of fnstenv is the lowest. It is interesting that the throughput of benign data is the lowest among all. The reason is that the benign data rarely contain sequence of NOP opcode, thus it rarely detected and terminated early. Thus, it takes more time for heuristic 1 to finish a stream.

Heuristic 2:

Self-contained

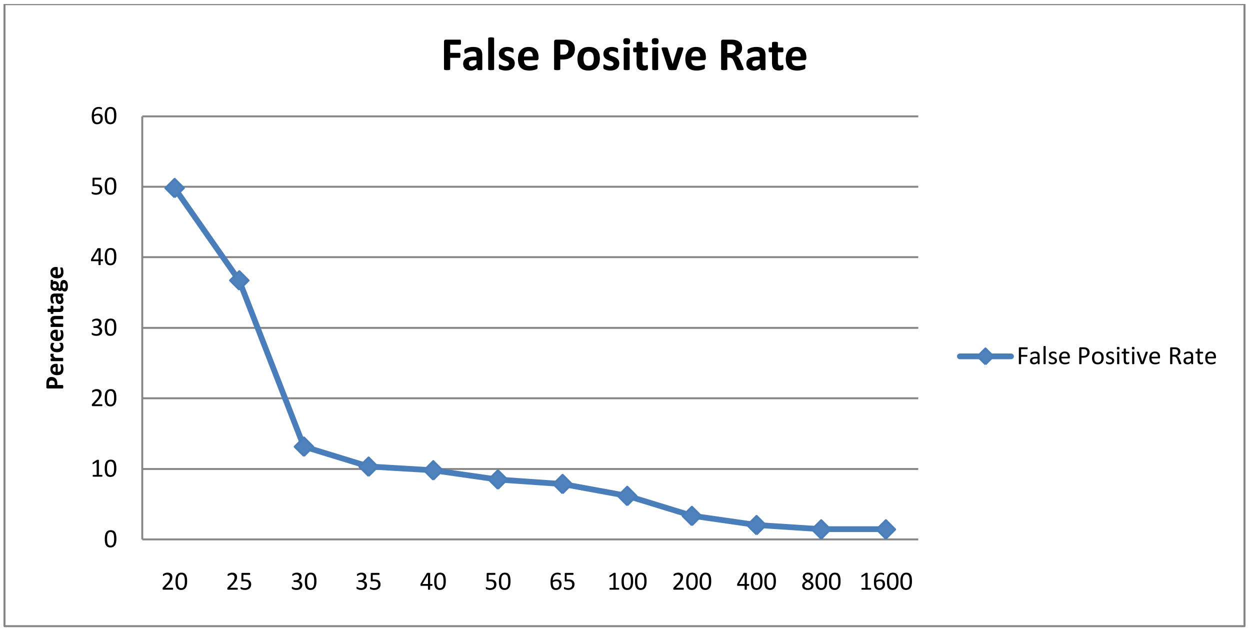


Figure 3: SC_XT = 2048

SC_PRT	False Positive RATE	Total shell code
20	49.82	21853
25	36.73	16114
30	13.16	5775
35	10.33	4531
40	9.81	4305
50	8.51	3734
65	7.87	3454
100	6.18	2713
200	3.34	1466
400	2.04	897
800	1.44	633
1600	1.44	633

We consider 10.33% of false positive rate is acceptable, thus, we choose 35 as SC_SRT. We then investigate how SC_PRT affect the detection rate.

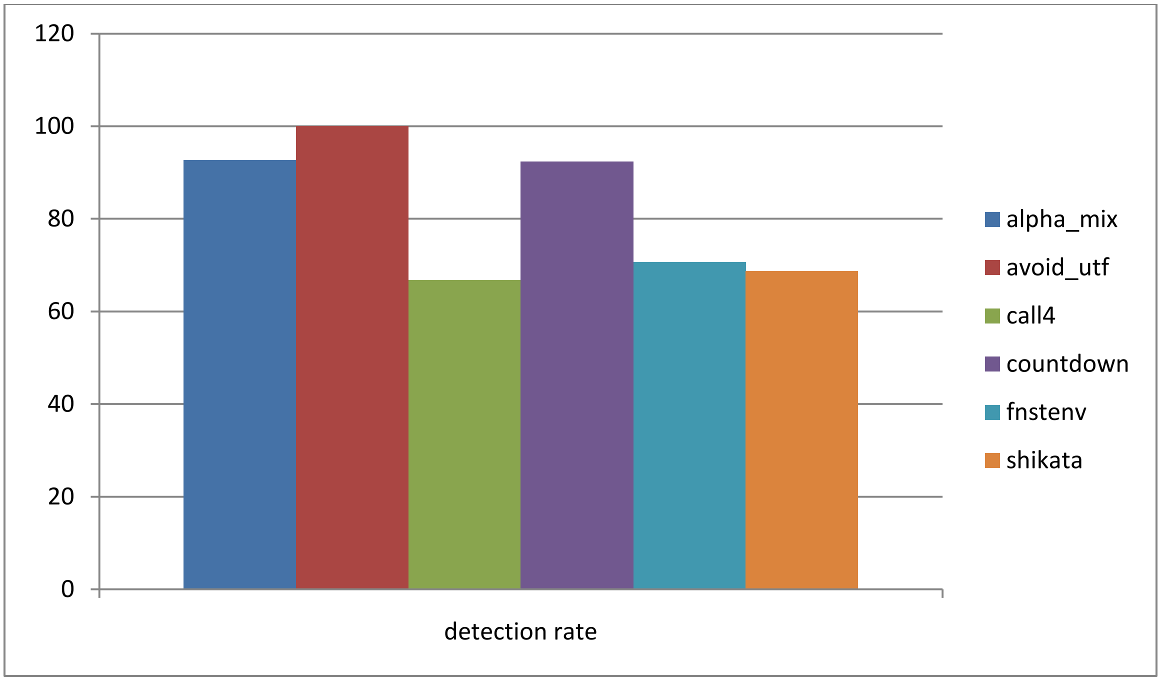


Figure 4: SC_XT = 2048, SC_PRT = 35

	alpha_mix	avoid_utf	call4	countdown	fnstenv	shikata
detection rate	92.76	100	66.8	92.31	70.69	68.75

The detection rate on alpha_mix, avoid_utf, and countdown are high. The possible reason is that heuristic 2 detects arithmetic and jump opcode as evidence of shell code. Alpha_mix and avoid_utf have high percentage of arithmetic opcode. Regarding countdown, the detection rate does not correspond to the code type distribution. We suspect the result may not reveal much statistically meaningful since avoid_utf has only 91 streams.

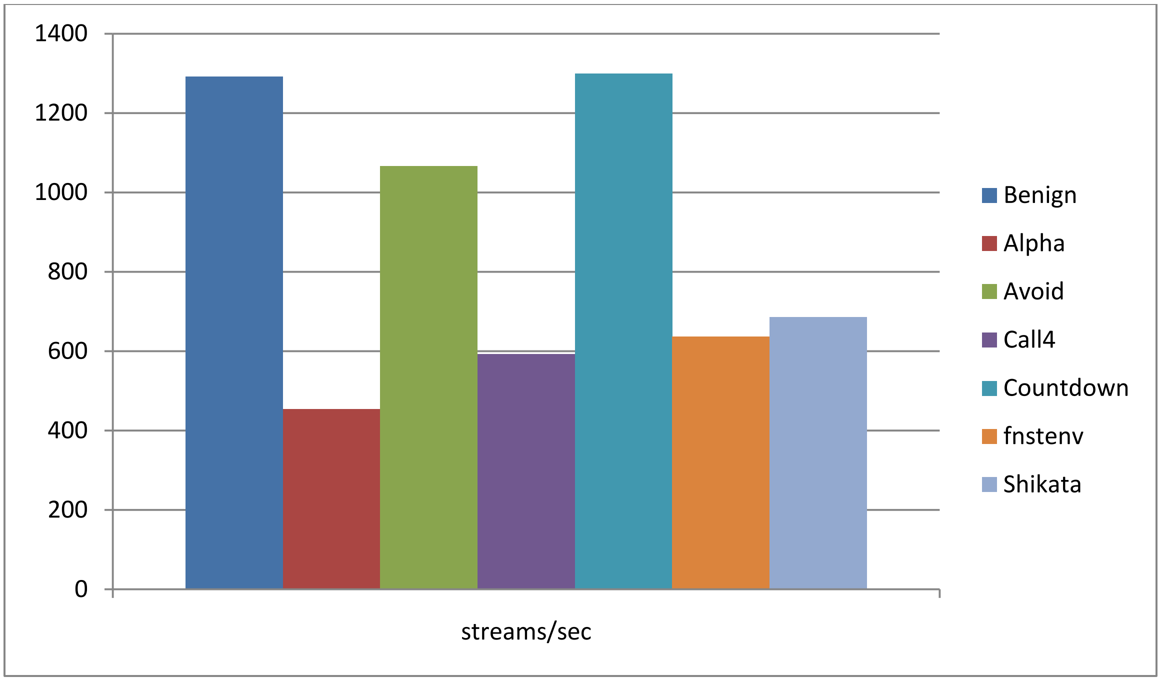


Figure 5: SC_PRT = 19, SC_XT = 2048

On the contrary to heuristic 1, benign data has high throughput here. The possible reason is that heuristic 2 observes arithmetic opcode as an evidence of shellcode. However, benign data has high percentage of arithmetic (~38%) and jump opcode (18%), please see code distribution in Part 2. Thus, some streams in benign data are “detected” and terminated early. Avoid_utf8 has relatively high percentage of arithmetic opcode and Countdown has high percentage of jump opcode.

Heuristic 3:

Non-self-contained

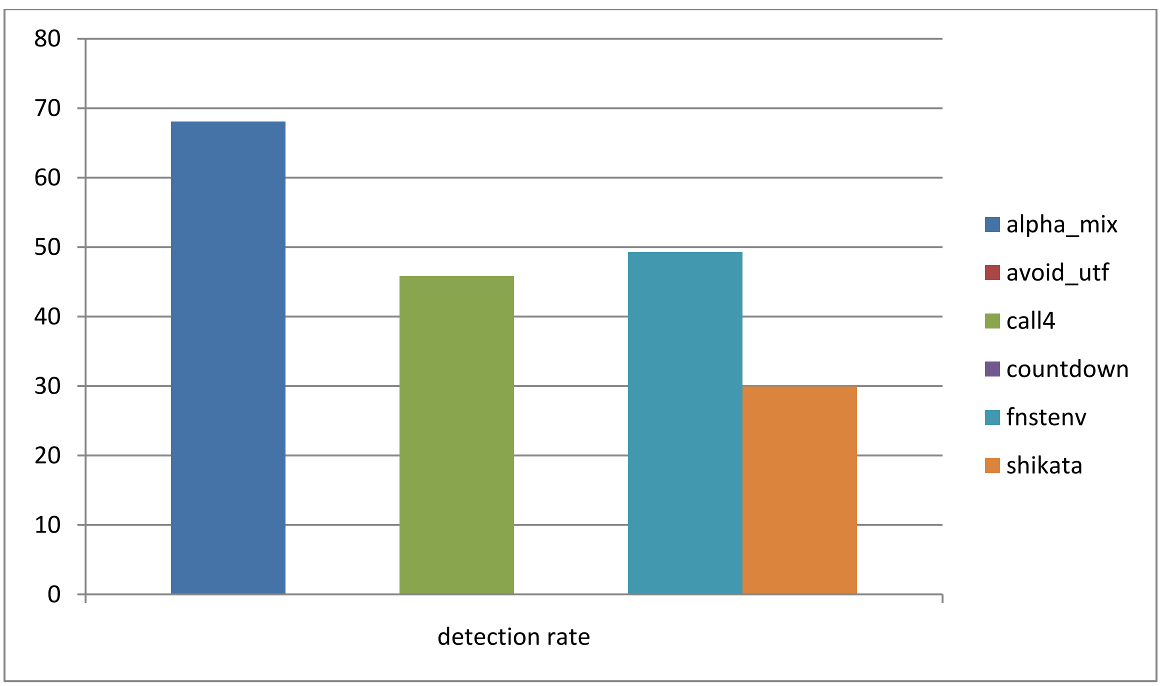
*Since it takes too much time running benign data, we generate a "shrink" version of benign data: We take first 200,000 packets of the original benign data.

Using the default parameters: NSC_MIN_WRITES = 8, NSC_MIN_WX = 14, NSC_XT = 10000, heuristic 3 gives 0 false positive rate but very poor detection rate.

We found that the false positive rate keeps really low (0.01%) when we decrease the NSC_MIN_WRITES and NSC_XT. We then optimize the detection rate by setting NSC_MIN_WRITES = 2 and NSC_XT = 2.

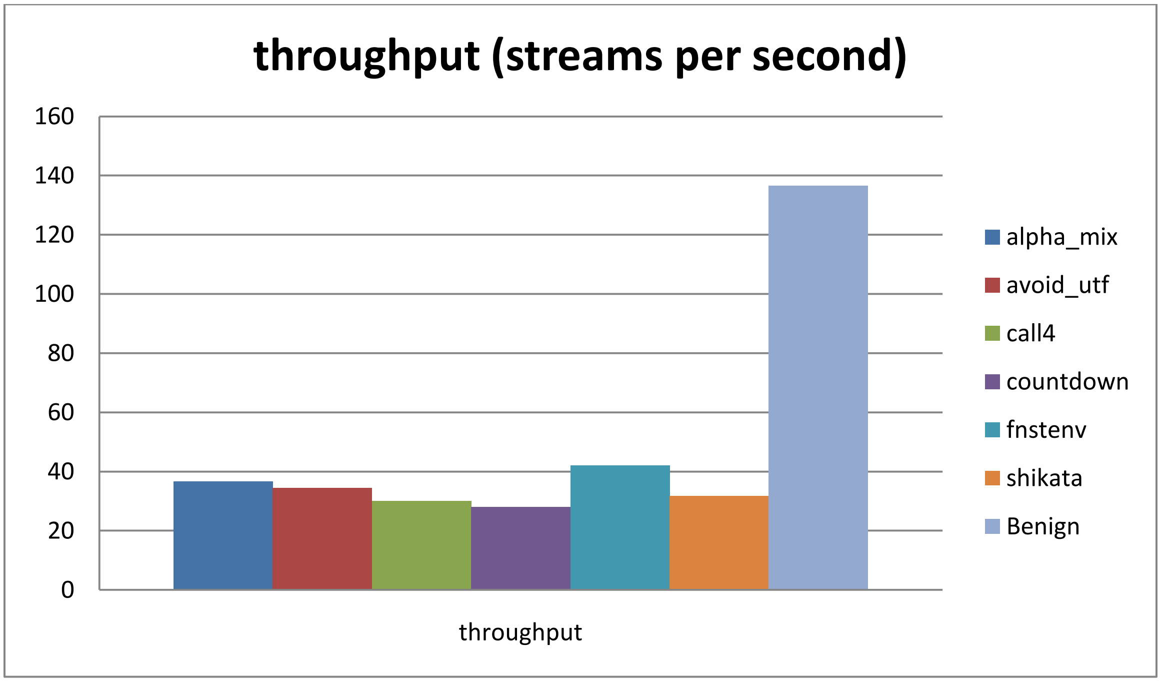
* Since there are very few streams in avoid_utf and countdown, there are no streams detected.

The detection rate is too low. The possible reason is that many streams are short, therefore it is rare that memory write and wx-instruction can be detected within such short stream.



	alpha_mix	avoid_utf	call4	countdown	fnstenv	shikata
detection rate	68.09	0	45.82	0	49.28	29.95

To our surprise, the throughput of benign data is the lowest. We thought that a benign stream terminates until the number of executions reaches NSC_XT. The possible reason is that each stream of benign data is short; thus, it terminates very early compared to shell code.

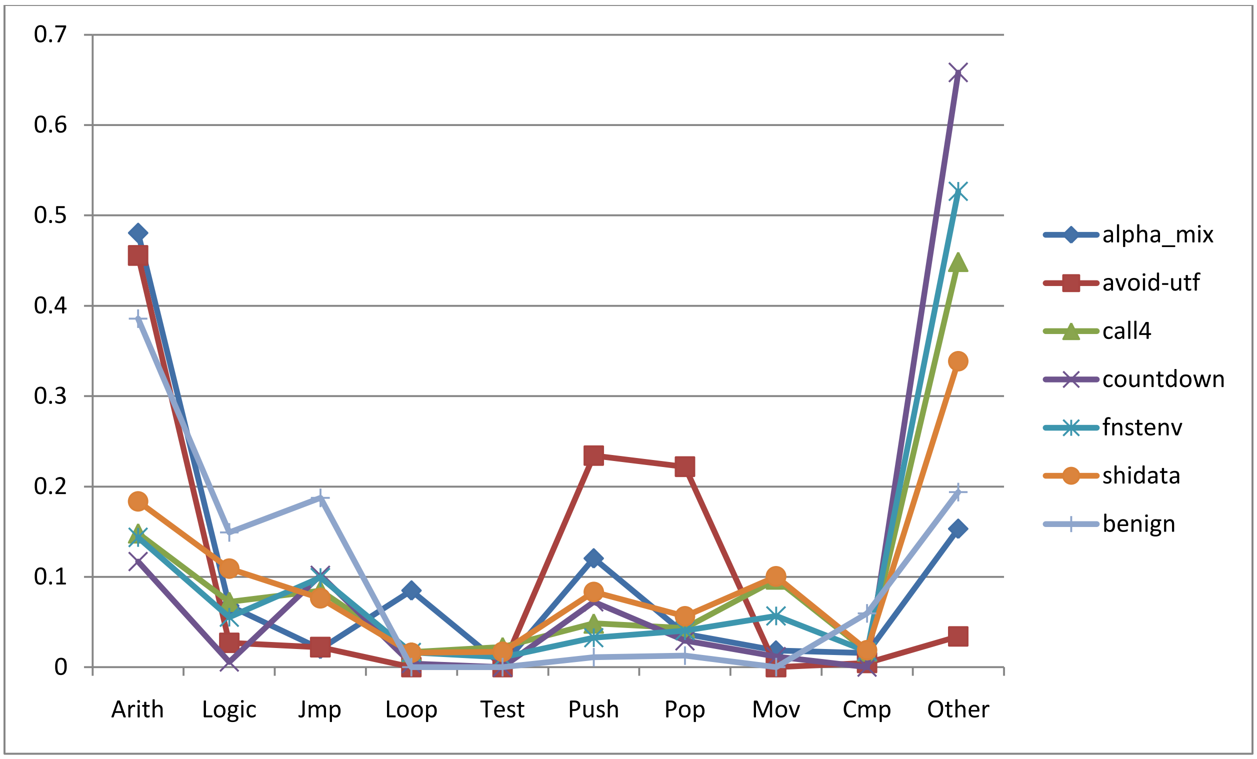


	alpha_mix	avoid_utf	call4	countdown	fnstenv	shikata	Benign
throughput	36.714976	34.408602	30.086669	28	42.03513	31.73554	136.4703
streams	608	32	729	91	694	384	43867
sec	16.56	0.93	24.23	3.25	16.51	12.1	321.44

Part 2:

Classifying the polymorphic engine

We try to derive the features from code type distribution. We believed that arithmetic, logic, jump, loop, test, push, pop, move, compare are 9 most frequent code type. Thus, we use these code types and other as features. Some features do separate one polymorphic engine from others. For example, push and pop types distinguish avoid_utf from otherengines, while arithmetic distinguishes alpha_mix and avoid_utf from others.



	alpha_mix	avoid_utf	call4	countdown	fnstenv	shikata	Benign
Arith	0.480588583	0.455637167	0.148220308	0.116924	0.143653393	0.183599167	0.385747
Logic	0.067947694	0.0270895	0.072504154	0.005894	0.055297179	0.109140833	0.149203
Jmp	0.020196083	0.022201833	0.084904923	0.1016665	0.099282214	0.076376167	0.187444
Loop	0.084856694	0	0.016795962	0.003894	0.016365357	0.0159935	8.06E-06
Test	0.001884361	0	0.022352885	0	0.011026036	0.017189667	1.02E-05
Push	0.120408944	0.234119167	0.048461846	0.072409	0.032673321	0.083331833	0.011144
Pop	0.036789306	0.221959833	0.043450154	0.0293635	0.040504036	0.056406167	0.012972
Mov	0.018409361	0	0.097151731	0.011682	0.056853214	0.100482667	0.000101
Cmp	0.015755556	0.004829167	0.0174965	0	0.017556464	0.018873	0.059493
Other	0.153163361	0.034163	0.448662038	0.6581665	0.526789071	0.338608	0.193878

We then use minimum distance classifier to determine which polymorphic engine best matches the unknown engine. In addition, we set the maximum distance bound. If the error exceeds the bound, we classify the engine as unknown.

If the payloads were encoded quite "diversely", the classifier may not work very well in the given data set. It does not distinguish each payload well. Since we do not know how exactly each payload is encoded, we may not evaluate this point. Most of the payload is classified as alpha_mix and many are classified as Shikata.

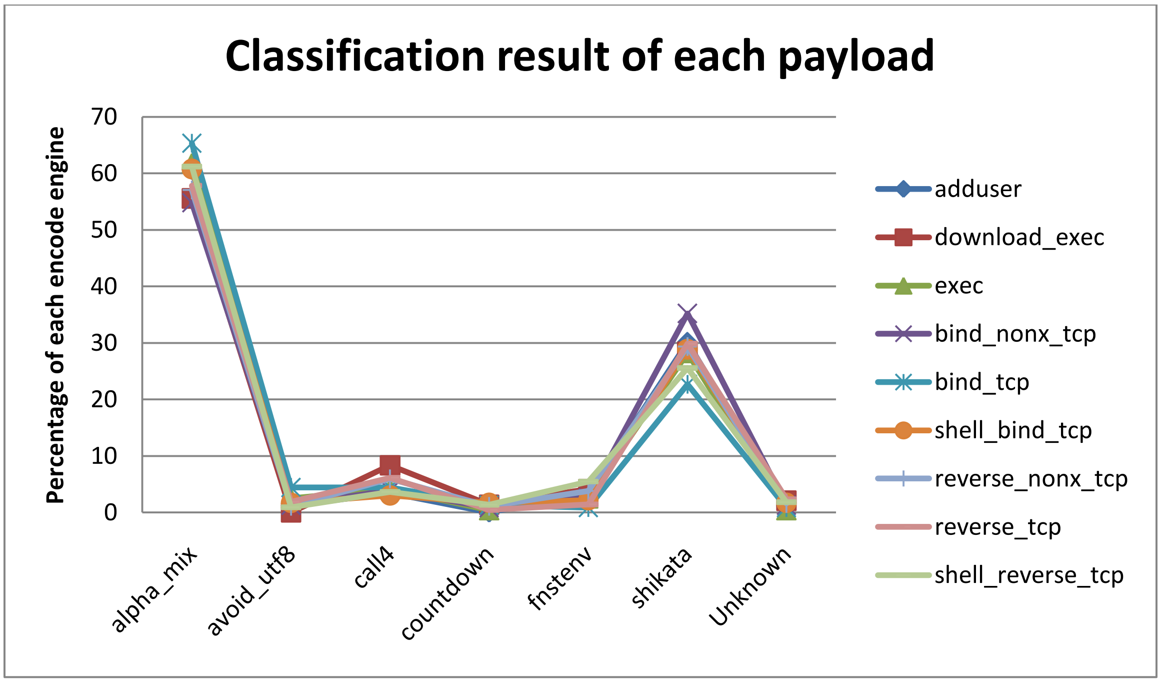


Figure 6: Evaluation of classifier

	adduser	download_ exec	exec	bind_ nonx_tcp	bind_tcp	shell_ bind_tcp	reverse_ nonx_tcp	reverse_ tcp	shell_ reverse_tcp
alpha_mix	60.62	55.56	61.83	54.78	65.33	60.78	57.07	57.82	61.19
avoid_utf8	1.16	0	2.49	1.3	4.44	1.72	1.09	1.9	0.91
call4	3.47	8.33	4.15	3.91	4.44	3.02	5.98	6.16	3.65
countdown	0	1.39	0.41	0.87	1.33	1.72	1.09	0.47	1.37
fnstenv	3.47	4.17	2.49	2.61	0.89	2.16	3.8	1.42	5.48
shikata	30.12	28.47	28.22	35.22	22.67	28.88	29.35	29.86	25.57
Unknown	1.16	2.08	0.41	1.3	0.89	1.72	1.63	2.37	1.83

Part 3:

Determining the function family

Most of the shell functions are classified as unknown. It seems weird. We examine the shell code and figure out that most of the shell codes terminate shortly after the offset, so that they do not reach the api call. Streams that do not reach api call are classified as unknown. This is the reason that the unknown percentage is so high.

For avoid_utf and countdown, since there are very few streams, there are not any streams that reach the api call.

	AddUser	FTPExec	HTTPExec	ConnectExec	BindShell	BindExec	Unknown
alpha_mix	0	0.53	0	1.006	0	0.89	97.52
avoid_utf	0	0	0	0	0	0	100
call4	0	1.23	0.21	0.41	0	1.64	96.51
countdown	0	0	0	0	0	0	0
fnstenv	0	0.2	0	0.61	0	0.41	98.78
shikata	0	0.76	0	0	0	0.38	98.86