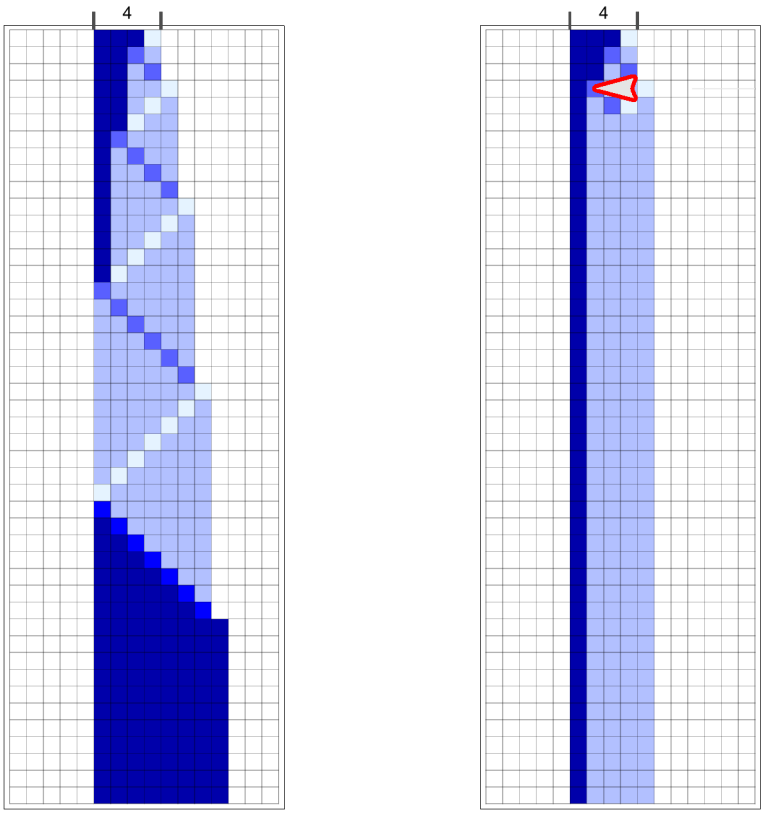
A Minimal Model for Computer Security

Willem Nielsen

Out[ ]=



Motivation & Goal

I was lucky enough to work with Stephen Wolfram (creator of Mathematica and Wolfram|Alpha) on his recent research. His work is centered on minimal computational models, in contrast to most theoretical scientists who use mathematical models.

While I was working with him, he suggested to me to use these computational systems to try to model the high-level aspects of computer security. I thought this project would be a good opportunity to do this.

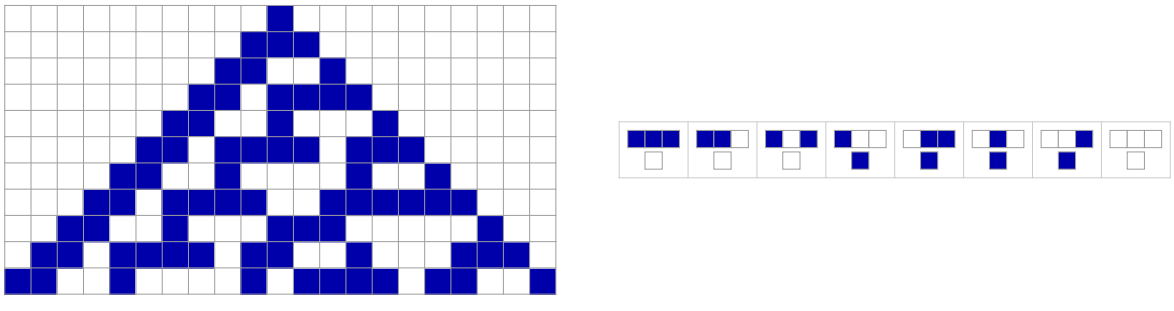
In this article, I will use cellular automata, one of Stephen's favorite computational systems, as a minimal model for engineered software programs. Then, I will look at what happens when you perturb these programs, as a model for an attacker compromising the system.

Background

So first of all, what is a cellular automata? (or a CA as I will refer to it)

A CA has an initial condition in the form of a row of discrete cells, shown here as the first row of the pattern on the left. The rules of a CA, shown here on the right, then dictate how the pattern evolves from one row to the next:

Out[41]=

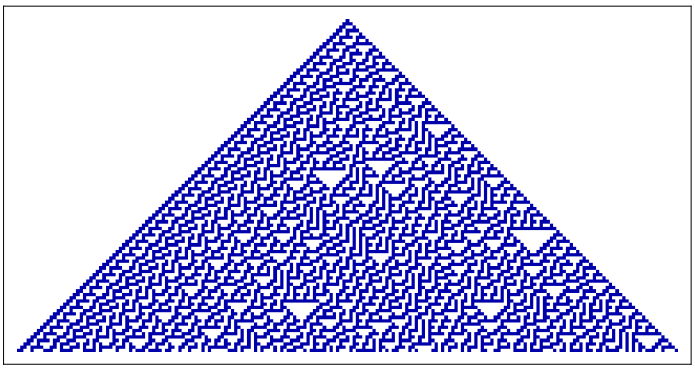


So for instance, the initial condition here contains the sequence white, white, blue. The output of the rule for that sequence is blue, which is then placed in the next row. Moving one step to the right we have a sequence of white, blue, white. The output rule for that sequence is also blue, which is placed in

the next row one cell to the right. This updating procedure is repeated for all the subsequences at each row down the page.

This particular CA, rule 30, as its called, is important because its one of the simplest CAs that can generate complex behavior. So, starting from this simple initial condition of a single black cell, and following these eight rules, the program generates a pattern that is completely random:

Out[43]=



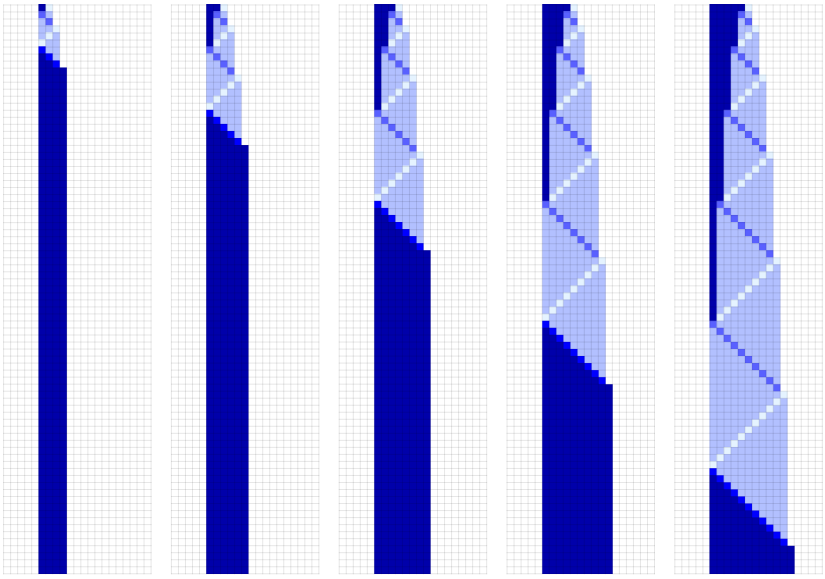
Coincidentally, this rule is used as a cryptographic algorithm for generating key-pairs. And in fact, one of the directions of future work that were interested in is using it to engineer a cryptocurrency.

But I'm not going to discuss that here, instead, in light of trying to model the challenges of computer security, we're going to look at how we can use CAs as a minimal model for a software system.

A Minimal Model for a Software System

Rule 30, while complex, doesn't seem to be doing any "useful computation" of the kind we're used to seeing in software systems. But, Wolfram, in his book *New Kind of Science*, actually presented some CAs that can do useful computation. One example is the CA shown below, which can be thought of as a procedure for doubling its input:

Out[ ]=



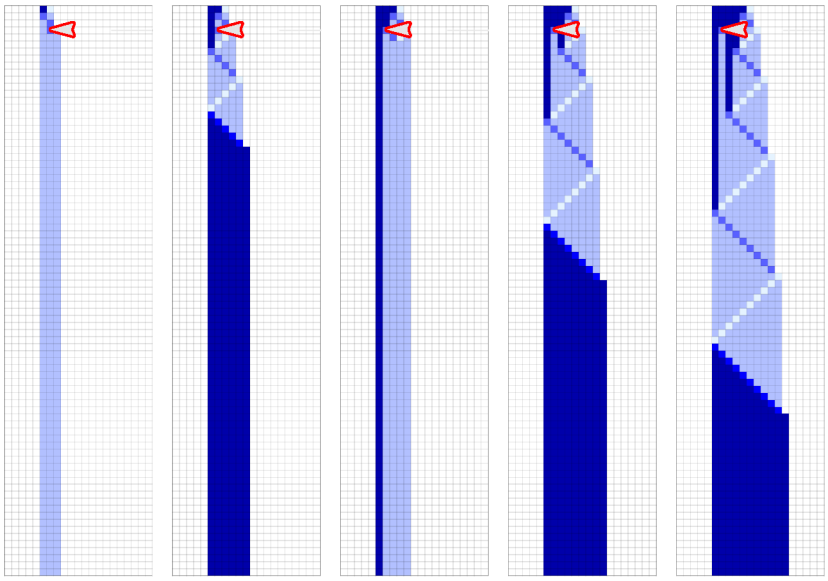
The initial row, contains a string of two colored cells: one dark blue and one light blue. After running the rules for some number of steps, the final row contains a string of four colored cells, twice that of the initial row. The next panel has an initial condition with a string of three colored cells: two dark blue and one light blue. After running the same rules on this initial condition, the final row contains a string of *six* colored cells, twice the number from the initial row.

We can see from the pattern this calculation is performed in a very sequential manner, much like the way engineered software systems perform computation.

Modeling an Attacker

Now that we have this minimal model for an engineered software program, we can model an attacker by perturbing cells in the pattern, and observe the resulting behavior.

Out[ ]=



In some cases, like the second panel, the perturbation makes basically no change to the system, almost like changing an arbitrary print statement in the code of the program.

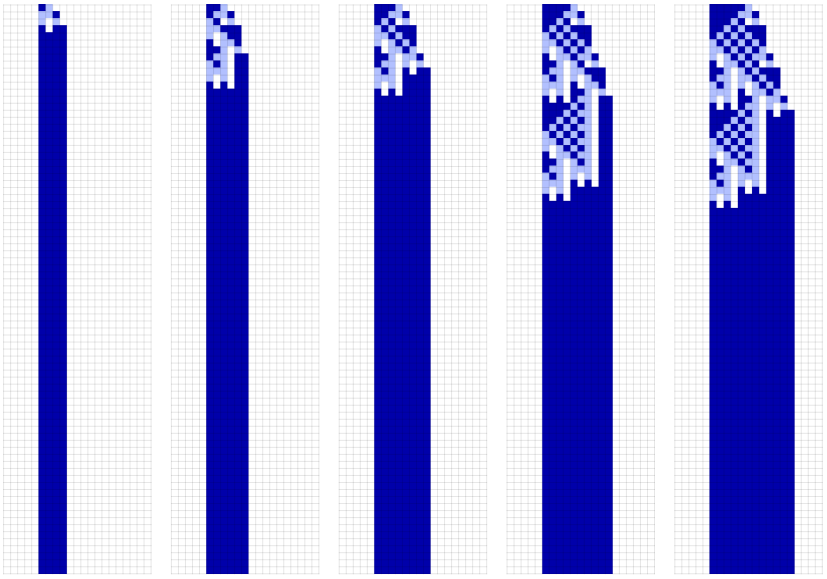
In the first and third panel, the perturbation, results in some trivial behavior, almost like killing the program immediately. In these cases, in contrast to the first panel, the resulting output is incorrect. In the fourth and fifth panel, the error is much more subtle, only slightly altering the behavior of the system, kind of like running a for-loop one too many times.

One thing to note, is with this CA at least, the resulting perturbations don't lead to whole new classes of behavior, limiting the ability of an attacker to compromise the system. But what about other CAs?

A More Efficient Procedure

We can look at at other CAs perform the doubling procedure "more efficiently", i.e. in fewer steps:

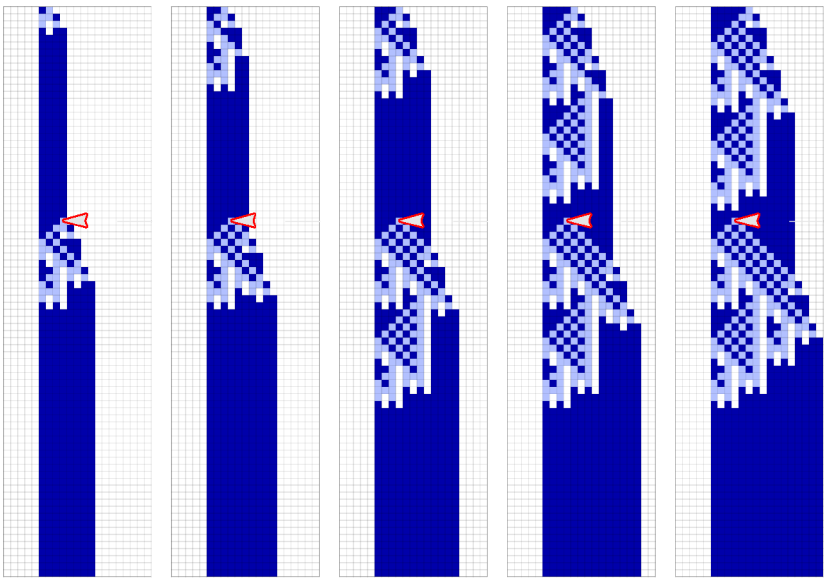
Out[ ]=



This CA, instead of working in a sequential manner, essentially parallelizes the computation, resulting in the algorithm taking fewer steps. So what happens if we perturb this CA?

We can see that an intelligent attacker using the right perturbation has the ability to change the behavior faster than in the previous CA. With this particular perturbation, or compromise, the attacker can get the program to give an output that is too large for each input:

Out[ ]=



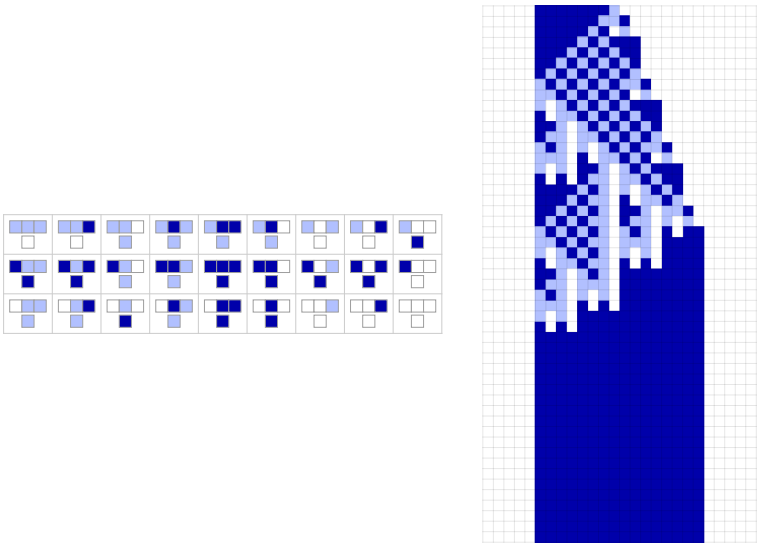
One way to think about this is, when using a more sophisticated procedure, the program becomes more malleable to attackers.

A Larger Attack Surface

Along similar lines, we can look at what happens if the attacker is able to perturb not just cells in the pattern but an entire rule, effectively increases the surface of attack.

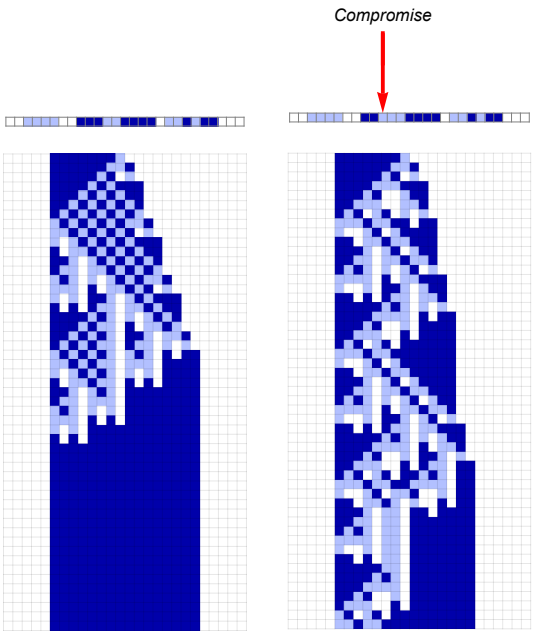
So here is the pattern and rules for our more efficient CA:

Out[ ]=



Now, on the top we arrange the output of all our rules in a row, and change a single rule in that row from dark blue to light blue, shown by the red arrow:

Out[ ]=

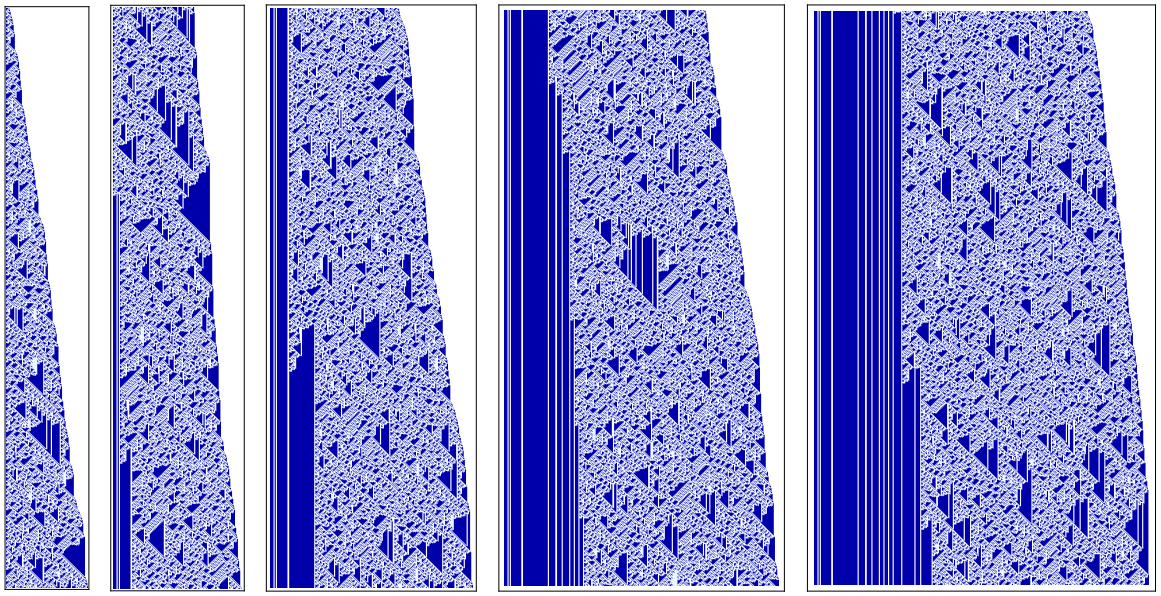


And as you may expect, the resulting change to the pattern is much more drastic.

And in fact, if we run this compromised rule for many more steps, we can see that its now capable of an

almost infinite number of new behaviors:

Out[44]=



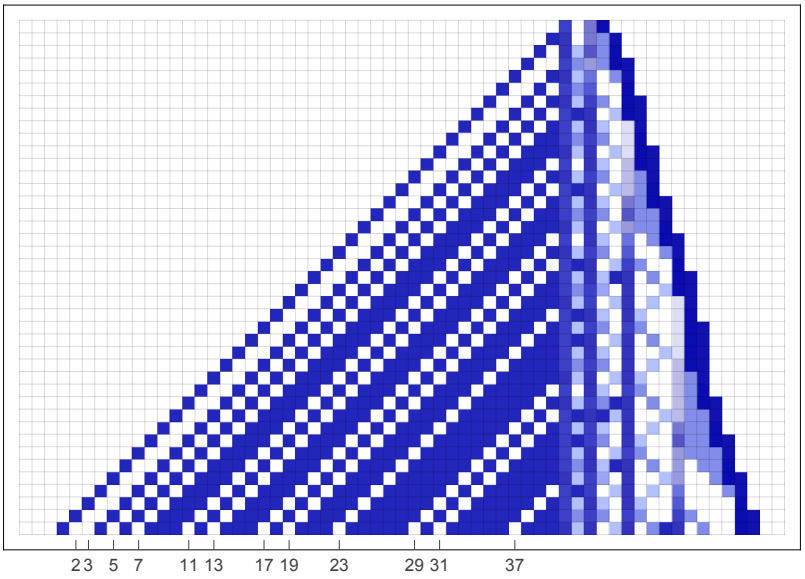
Conclusion

One thing I'I think we've seen from this course is just how difficult computer security is. And I believe the reason can be seen from the experiments that I've highlighted here. Namely, as Stephen Wolfram discovered with rule-30 back in early 1980s, even the simplest programs are capable of arbitrarily complex behavior. And therefore as engineers trying to limit what an attacker can do to a software system, we are essentially fighting what Stephen calls computational irreducibility.

Future work

A common reaction to these minimal models is to say that they are nothing like the practical systems used in the real world. But actually, as we've seen, even the simplest of cellular automata are capable of arbitrary complexity. And in fact, it has been shown that cellular automata are what's called Turing-complete. In other words, they can be made to emulate any program of arbitrary sophistication. As an example of this, Wolfram has constructed a CA that can compute the prime numbers:

Out[ ]=



So one thing that we can look at in the future is what happens when we perturb these CAs running more complicated procedures?

Overall, in this article, I've hoped to have convinced the reader that using these simple computational models, we can study not just individual algorithms and methods of attack, but also look at computer security at a higher level and draw conclusions about the field as whole. I think this is a fascinating line of research and one that I plan on continuing after this course. Thank you for reading and thank you for a fun semester!